09/506,502

APPENDIX A

README.gf

```
Notation: <dir>    - path to this directory

This is a brief list of the files in this directory.

Miscellaneous:
    README.gf               this file
    gf.tie                  a copy of the source tie file
    default-params                default param file to configure software tools
    gf-params               param file to configure software tools for gf.tie

Native C Support:
    cstub-gf.c              functions for the new instructions
    cstub-gf-ref.c                functions generated from "reference"
    BR.h                    support for BR register file

Design Compiler Synthesis:
    gf.v                    Verilog source file
    gf_check.dcsh                 Syntax check generated verilog
    gf.dcsh                 Top-level Design Compiler synthesis script
    Xtensa_cons_generic.dc        supporting script
    Xtensa_prim.dc                supporting script
    TIE_opt.dc                    supporting script
    xmTIE_cons.dc           supporting script
    prim.v                  supporting Verilog source file

Verysys Verification:
    verysys                 subdirectory supporting Verysys verification
    verysys/verify_sem.v          Verilog source generated from semantics
    verysys/verify_ref.v          Verilog source generated from reference

Xtensa tool support:
    libisa-gf.so            dynamically linked library for xt-gcc
    libiss-gf.so            dynamically linked library for xt-run
    xtensa-gf.h             macro definitions of new instructions

Unknown:
    gf_test.v


To compile your application in native mode:
    - include cstub-gf.c in your application
    - compile your application using your native c compiler (e.g., gcc)

    The new TIE instructions are replaced with equivalent C code.

    If you define add "-DTIE_DEBUG" to the C compile, the function names for
    the translated TIE instructions will be prefixed with "TIE_".  Using this
    method, you can check the TIE description against hand-written C functions
    for the new instructions.  Refer for the application note for more
    details.

To compile your application for Xtensa:
    - add "--xtensa-params=<dir>" to the command line or add the environment
      variable 'XTENSA_PARAMS=<dir>; export XTENSA_PARAMS"
    - compile your application using xt-gcc
```

To estimate the impact of your TIE description on Xtensa speed:
- Setup your shell environment to run Synopsys Design Compiler
- Modify gf.dcsh to fill in your technology information
- Run dc_shell with script gf.dcsh, e.g.,
  "dc_shell -f gf.dcsh >& dc.out &"
- Inspect the synthesis results. Look in report section of the output file "dc.out". If there is any timing violation, the Xtensa speed will be impacted, roughly by the violation amount. The area report section will give you the area of your tie instruction block.

To compare reference designs against semantic designs using Verysys:
- "cd <dir>/verysys; make"

Notes for v1.5 user:
- cstub-gf.c can be included for xt-gcc compiles; it will be ignored

Note for v1.1 user:
- no need to regenerate this development kits from the Web.
- no need to include <machine/Customer.h> anymore

gf.tie

```
opcode      GFADD8      op2=4'b0000 CUST0
opcode      GFMULX8     op2=4'b0001 CUST0
opcode      GFRWMOD8 op2=4'b0010    CUST0
opcode      GFADD8I     op2=4'b0100 CUST0
opcode      LGF8.I      r=4'b0000   LSCI
opcode      SGF8.I      r=4'b0001   LSCI
opcode      LGF8.IU     r=4'b0010   LSCI
opcode      SGF8.IU     r=4'b0011   LSCI
opcode      LGF8.X      op2=4'b0000 LSCX
opcode      SGF8.X      op2=4'b0001 LSCX
opcode      LGF8.XU     op2=4'b0010 LSCX
opcode      SGF8.XU     op2=4'b0011 LSCX
state gfmod 8
user_register 0 { gfmod }
regfile gf 8 16 g
operand gr r { gf[r] }
operand gs s { gf[s] }
operand gt t { gf[t] }
operand imm4 t { t } { imm4 }

interface VAddr          32    core   out
interface LSSize          5    core   out
interface MemDataIn8      8    core   in
interface MemDataOut8     8    core   out

iclass gfrrr { GFADD8 } {out gr, in gs, in gt} {} {}
iclass gfrri { GFADD8I } {out gr, in gs, in imm4} {} {}
iclass gfrr  { GFMULX8 } {out gr, in gs} {in gfmod} {}
iclass gfr  { GFRWMOD8 } {inout gt} {inout gfmod} {}
iclass gfloadi { LGF8.I } { out gt, in ars, in imm8} {} {
```

```
    out LSSize, out VAddr,
    in MemDataIn8 }
  iclass gfstorei { SGF8.I } { in gt, in ars, in imm8} {} {
    out LSSize, out VAddr,
    out MemDataOut8 }
  iclass gfloadiu { LGF8.IU } { out gt, inout ars, in imm8} {} {
    out LSSize, out VAddr,
    in MemDataIn8 }
  iclass gfstoreiu { SGF8.IU } { in gt, inout ars, in imm8} {} {
    out LSSize, out VAddr,
    out MemDataOut8 }
  iclass gfloadx { LGF8.X } { out gr, in ars, in art} {} {
    out LSSize, out VAddr,
    in MemDataIn8 }
  iclass gfstorex { SGF8.X } { in gr, in ars, in art} {} {
    out LSSize, out VAddr,
    out MemDataOut8 }
  iclass gfloadxu { LGF8.XU } { out gr, inout ars, in art} {} {
    out LSSize, out VAddr,
    in MemDataIn8 }
  iclass gfstorexu { SGF8.XU } { in gr, inout ars, in art} {} {
    out LSSize, out VAddr,
    out MemDataOut8 }
  semantic gf1 { GFADD8 } {
    assign gr = gs ^ gt;
  }
  semantic gf4 { GFADD8I } {
    assign gr = gs ^ imm4;
  }
  semantic gf2 { GFMULX8 } {
    assign gr = gs[7] ? ({gs[6:0],1'b0} ^ gfmod) : {gs[6:0],1'b0};
  }
  semantic gf3 { GFRWMOD8 } {
    wire [7:0] t1 = gt;
    wire [7:0] t2 = gfmod;
    assign gfmod = t1;
    assign gt = t2;
  }
  semantic lgf { LGF8.I, LGF8.IU, LGF8.X, LGF8.XU } {
    wire indexed = LGF8.X|LGF8.XU;
    assign LSSize = 1;
    assign VAddr = ars + (indexed ? art : imm8);
    assign gt = MemDataIn8;
    assign gr = MemDataIn8;
    assign ars = VAddr;
  }
  semantic sgf { SGF8.I, SGF8.IU, SGF8.X, SGF8.XU } {
    wire indexed = SGF8.X|SGF8.XU;
    assign LSSize = 1;
    assign VAddr = ars + (indexed ? art : imm8);
    assign MemDataOut8 = SGF8.X|SGF8.XU ? gr : gt;
    assign ars = VAddr;
  }

reference GFADD8 {
  assign gr = gs ^ gt;
}
```

```
reference GFADD8I {
  assign gr = gs ^ imm4;
}
reference GFMULX8 {
  assign gr = gs[7] ? ({gs[6:0],1'b0} ^ gfmod) : {gs[6:0],1'b0};
}
reference GFRWMOD8 {
  wire [7:0] t1 = gt;
  wire [7:0] t2 = gfmod;
  assign gfmod = t1;
  assign gt = t2;
}
reference LGF8.I {
  assign LSSize = 1;
  assign VAddr = ars + imm8;
  assign gt = MemDataIn8;
}
reference LGF8.IU {
  assign LSSize = 1;
  assign VAddr = ars + imm8;
  assign gt = MemDataIn8;
  assign ars = VAddr;
}
reference LGF8.X {
  assign LSSize = 1;
  assign VAddr = ars + art;
  assign gr = MemDataIn8;
  assign ars = VAddr;
}
reference LGF8.XU {
  assign LSSize = 1;
  assign VAddr = ars + art;
  assign gr = MemDataIn8;
  assign ars = VAddr;
}
reference SGF8.I {
  assign LSSize = 1;
  assign VAddr = ars + imm8;
  assign MemDataOut8 = gt;
}
reference SGF8.IU {
  assign LSSize = 1;
  assign VAddr = ars + imm8;
  assign MemDataOut8 = gt;
  assign ars = VAddr;
}
reference SGF8.X {
  assign LSSize = 1;
  assign VAddr = ars + art;
  assign MemDataOut8 = gr;
}
reference SGF8.XU {
  assign LSSize = 1;
  assign VAddr = ars + art;
  assign MemDataOut8 = gr;
  assign ars = VAddr;
}
```

```
ctype gf8 8 8 gf
proto gf8_loadi {out gf8 t, in gf8* s, in immediate o} {} {
      LGF8.I      t, s, o;
}
proto gf8_storei {in gf8 t, in gf8* s, in immediate o} {} {
      SGF8.I      t, s, o;
}
proto gf8_move {in gf8 r, in gf8 s} {} {
      GFADD8      r, s, 0;
}

schedule gfload { LGF8.I }
{
  use imm8 0;
  use ars 1;
  def gt 2;
}

schedule gfloadu { LGF8.IU }
{
  use imm8 0;
  use ars 1;
  def ars 1;
  def gt 2;
}

schedule gfloadx { LGF8.X }
{
  use ars 1;
  use art 1;
  def gr 2;
}

schedule gfloadxu { LGF8.XU }
{
  use ars 1;
  use art 1;
  def art 1;
  def gr 2;
}

synopsis GFADD8  "Galois Field 8-bit Add"
synopsis GFADD8I "Galois Field 8-bit Add Immediate"
synopsis GFMULX8 "Galois Field 8-bit Multiply by X"
synopsis GFRWMOD8 "Read/Write Galois Field Polynomial"
synopsis LGF8.I  "Load Galois Field Register Immediate"
synopsis LGF8.IU "Load Galois Field Register Immediate Update"
synopsis LGF8.X  "Load Galois Field Register Indexed"
synopsis LGF8.XU "Load Galois Field Register Indexed Update"
synopsis SGF8.I  "Store Galois Field Register Immediate"
synopsis SGF8.IU "Store Galois Field Register Immediate Update"
synopsis SGF8.X  "Store Galois Field Register Indexed"
synopsis SGF8.XU "Store Galois Field Register Indexed Update"

description GFADD8
"<P><CODE>GFADD8</CODE> performs a 8-bit Galois Field addition of the
```

```
contents of GF registers <CODE>gs</CODE> and <CODE>gt</CODE> and
writes the result to GF register <CODE>gr</CODE>.</P>"

description GFADD8I
"<P><CODE>GFADD8I</CODE> performs a 8-bit Galois Field addition of the
contents of GF register <CODE>gs</CODE> and a 4-bit immediate from
the <CODE>t</CODE> field and writes the result to GF register
<CODE>gr</CODE>.</P>"

description GFMULX8
"<P><CODE>GFMULX8</CODE> performs a 8-bit Galois Field multiplication
of the contents of GF register <CODE>gs</CODE> by <I>x</I> modulo
the polynomial in <CODE>gfmod</CODE>.  It writes the result to GF register
<CODE>gr</CODE>.</P>"

description GFRWMOD
"<P><CODE>GFRWMOD</CODE> reads and writes the <CODE>gfmod</CODE>
polynomial register.  GF register <CODE>gt</CODE> and <CODE>gfmod</CODE>
are read these are written to <CODE>gfmod</CODE> and <CODE>gt</CODE>.</P>"

description LGF8.I
"<P>
</P>"

description LGF8.IU
"<P>
</P>"

description LGF8.X
"<P>
</P>"

description LGF8.XU
"<P>
</P>"

description SGF8.I
"<P>
</P>"

description SGF8.IU
"<P>
</P>"

description SGF8.X
"<P>
</P>"

description SGF8.XU
"<P>
</P>"
```

default-params

```
isa-tie-dll=lib-i686-Linux/libisa-gf.so
iss-tie-dll=lib-i686-Linux/libiss-gf.so
```

```
cc-tie-dll=lib-i686-Linux/libcc-gf.so
xtensa-tie-header=xtensa-gf.h
```

gf-params

```
isa-tie-dll=lib-i686-Linux/libisa-gf.so
iss-tie-dll=lib-i686-Linux/libiss-gf.so
cc-tie-dll=lib-i686-Linux/libcc-gf.so
xtensa-tie-header=xtensa-gf.h
```

cstub-gf.c

```
#ifndef __XTENSA__
#ifdef TIE_DEBUG
#define gf8_loadi TIE_gf8_loadi
#define gf8_storei TIE_gf8_storei
#define gf8_move TIE_gf8_move
#define GFADD8 TIE_GFADD8
#define GFADD8I TIE_GFADD8I
#define GFMULX8 TIE_GFMULX8
#define GFRWMOD8 TIE_GFRWMOD8
#define LGF8_I TIE_LGF8_I
#define SGF8_I TIE_SGF8_I
#define LGF8_IU TIE_LGF8_IU
#define SGF8_IU TIE_SGF8_IU
#define LGF8_X TIE_LGF8_X
#define SGF8_X TIE_SGF8_X
#define LGF8_XU TIE_LGF8_XU
#define SGF8_XU TIE_SGF8_XU
#define RUR0 TIE_RUR0
#define WUR0 TIE_WUR0
#endif

#include <stdio.h>
#define LittleEndian    0
#define BigEndian 1
#define PIFReadDataBits 128
#define PIFWriteDataBits      128
#define IsaMemoryOrder  LittleEndian
#include "BR.h"
#include "LS.h"
#define BPW 32
#define WINDEX(_n) ((_n) / BPW)
#define BINDEX(_n) ((_n) % BPW)

typedef unsigned char Vb_t;
typedef unsigned short Vs_t;
typedef struct V1_s {unsigned data[1];} V1_t;
typedef struct V2_s {unsigned data[2];} V2_t;
typedef struct V4_s {unsigned data[4];} V4_t;

typedef Vb_t gf8;
```

```
static int tie_load_instruction = 0;

void
TieMemRead(unsigned *data, unsigned addr)
{
    unsigned char *mem;
    unsigned modulus, bytes, offset;
    int t, b0, b1, b2, b3;

    bytes = PIFReadDataBits / 8;
    modulus = bytes - 1;
    mem = (unsigned char *) (addr & ~modulus);
    offset = (unsigned char *) addr - mem;
    if (IsaMemoryOrder == LittleEndian) {
      for(t = 0; t < bytes/sizeof(int); t++) {
          b0 = mem[(offset++) & modulus];
          b1 = mem[(offset++) & modulus];
          b2 = mem[(offset++) & modulus];
          b3 = mem[(offset++) & modulus];
          data[t] = (b3 << 24) | (b2 << 16) | (b1 << 8) | b0;
      }
    } else {
      for(t = bytes/sizeof(int) - 1; t >= 0; t--) {
          b3 = mem[(offset++) & modulus];
          b2 = mem[(offset++) & modulus];
          b1 = mem[(offset++) & modulus];
          b0 = mem[(offset++) & modulus];
          data[t] = (b3 << 24) | (b2 << 16) | (b1 << 8) | b0;
      }
    }
}


void
TieMemWrite(unsigned addr, unsigned bytes, unsigned *data)
{
    unsigned char *mem;
    unsigned modulus, offset, w;
    int t;

    if (PIFWriteDataBits < bytes * 8) {
        fprintf(stderr, "Error: not configured to write %d bytes\n", bytes);
        exit(1);
    }

    modulus = bytes - 1;
    mem = (unsigned char *) (addr & ~modulus);
    if (IsaMemoryOrder == LittleEndian) {
      if (bytes == 1) {
          mem[0] = data[0] & 0xff;
      } else if (bytes == 2) {
          mem[0] = data[0] & 0xff;
          mem[1] = (data[0] >> 8) & 0xff;
      } else {
          offset = 0;
          for(t = 0; t < bytes/sizeof(int); t++) {
            w = data[t];
```

```
                mem[offset++] = w & 255;
                mem[offset++] = (w >> 8) & 255;
                mem[offset++] = (w >> 16) & 255;
                mem[offset++] = (w >> 24) & 255;
            }
        }
    } else {
        if (bytes == 1) {
            mem[0] = data[0] & 0xff;
        } else if (bytes == 2) {
            mem[1] = data[0] & 0xff;
            mem[0] = (data[0] >> 8) & 0xff;
        } else {
            offset = 0;
            for(t = bytes/sizeof(int) - 1; t >= 0; t--) {
              w = data[t];
              mem[offset++] = (w >> 24) & 255;
              mem[offset++] = (w >> 16) & 255;
              mem[offset++] = (w >> 8) & 255;
              mem[offset++] = w & 255;
            }
        }
    }
}
#define GetState(_s, _n) _s = _n
#define SetState(_n, _s) _n = _s

V1_t STATE_gfmod;

V1_t VAddr = {{0}};
V1_t VAddrBase = {{0}};
V1_t VAddrOffset = {{0}};
V1_t VAddrIndex = {{0}};
V1_t VAddrIn = {{0}};
V1_t LSSize = {{0}};
V1_t LSIndexed = {{0}};
V4_t MemDataIn128 = {{0,0,0,0}};
V2_t MemDataIn64 = {{0,0}};
V1_t MemDataIn32 = {{0}};
V1_t MemDataIn16 = {{0}};
V1_t MemDataIn8 = {{0}};
V4_t MemDataOut128 = {{0,0,0,0}};
V2_t MemDataOut64 = {{0,0}};
V1_t MemDataOut32 = {{0}};
V1_t MemDataOut16 = {{0}};
V1_t MemDataOut8 = {{0}};
V1_t Exception = {{0}};
V1_t ExcCause = {{0}};
V1_t CPEnable = {{0}};


void
VAddrIn_get(void)
{
    if (LSIndexed.data[0] != 0) {
        VAddrIn.data[0] = VAddrBase.data[0] + VAddrIndex.data[0];
    } else {
```

```
        VAddrIn.data[0] = VAddrBase.data[0] + VAddrOffset.data[0];
    }
}


void
MemDataIn128_get(void)
{
    unsigned data[4];

    if ((!tie_load_instruction) || (LSSize.data[0] != 16)) {
      return;
    }

    if (PIFReadDataBits < 128) {
      fprintf(stderr, "Error: not configured to read 16 bytes\n");
      exit(-1);
    }

    VAddrIn_get();
    TieMemRead(&data[0], VAddrIn.data[0]);
    MemDataIn128.data[0] = data[0];
    MemDataIn128.data[1] = data[1];
    MemDataIn128.data[2] = data[2];
    MemDataIn128.data[3] = data[3];
}


void
MemDataIn64_get(void)
{
    unsigned data[4];

    if ((!tie_load_instruction) || (LSSize.data[0] != 8)) {
      return;
    }

    if (PIFReadDataBits < 64) {
      fprintf(stderr, "Error: not configured to read 8 bytes\n");
      exit(-1);
    }

    VAddrIn_get();
    TieMemRead(&data[0], VAddrIn.data[0]);
    if (IsaMemoryOrder == LittleEndian) {
      MemDataIn64.data[0] = data[0];
      MemDataIn64.data[1] = data[1];
    } else if (PIFReadDataBits == 64) {
      MemDataIn64.data[0] = data[0];
      MemDataIn64.data[1] = data[1];
    } else {
      MemDataIn64.data[0] = data[2];
      MemDataIn64.data[1] = data[3];
    }
}
```

```
void
MemDataIn32_get(void)
{
    unsigned data[4];

    if ((!tie_load_instruction) || (LSSize.data[0] != 4)) {
      return;
    }

    if (PIFReadDataBits < 32) {
      fprintf(stderr, "Error: not configured to read 4 bytes\n");
      exit(-1);
    }

    VAddrIn_get();
    TieMemRead(&data[0], VAddrIn.data[0]);
    if (IsaMemoryOrder == LittleEndian) {
      MemDataIn32.data[0] = data[0];
    } else if (PIFReadDataBits == 32) {
      MemDataIn32.data[0] = data[0];
    } else if (PIFReadDataBits == 64) {
      MemDataIn32.data[0] = data[1];
    } else {
      MemDataIn32.data[0] = data[3];
    }
}


void
MemDataIn16_get(void)
{
    unsigned data[4];

    if ((!tie_load_instruction) || (LSSize.data[0] != 2)) {
      return;
    }

    if (PIFReadDataBits < 16) {
      fprintf(stderr, "Error: not configured to read 2 bytes\n");
      exit(-1);
    }

    VAddrIn_get();
    TieMemRead(&data[0], VAddrIn.data[0]);
    if (IsaMemoryOrder == LittleEndian) {
      MemDataIn16.data[0] = data[0] & 0xffff;
    } else if (PIFReadDataBits == 32) {
      MemDataIn16.data[0] = data[0] >> 16;
    } else if (PIFReadDataBits == 64) {
      MemDataIn16.data[0] = data[1] >> 16;
    } else {
      MemDataIn16.data[0] = data[3] >> 16;
    }
}


void
```

```
MemDataIn8_get(void)
{
    unsigned data[4];

    if ((!tie_load_instruction) || (LSSize.data[0] != 1)) {
      return;
    }

    if (PIFReadDataBits < 8) {
      fprintf(stderr, "Error: not configured to read 1 byte\n");
      exit(-1);
    }

    VAddrIn_get();
    TieMemRead(&data[0], VAddrIn.data[0]);
    if (IsaMemoryOrder == LittleEndian) {
      MemDataIn8.data[0] = data[0] & 0xff;
    } else if (PIFReadDataBits == 32) {
      MemDataIn8.data[0] = data[0] >> 24;
    } else if (PIFReadDataBits == 64) {
      MemDataIn8.data[0] = data[1] >> 24;
    } else {
      MemDataIn8.data[0] = data[3] >> 24;
    }
}


void
MemDataOut128_set(void)
{
    if (LSSize.data[0] != 16) {
      return;
    }

    VAddrIn_get();
    TieMemWrite(VAddrIn.data[0] & ~0xf, 16, &MemDataOut128.data[0]);
}


void
MemDataOut64_set(void)
{
    if (LSSize.data[0] != 8) {
      return;
    }

    VAddrIn_get();
    TieMemWrite(VAddrIn.data[0] & ~0x7, 8, &MemDataOut64.data[0]);
}


void
MemDataOut32_set(void)
{
    if (LSSize.data[0] != 4) {
      return;
    }
```

```
    VAddrIn_get();
    TieMemWrite(VAddrIn.data[0] & ~0x3, 4, &MemDataOut32.data[0]);
}


void
MemDataOut16_set(void)
{
    if (LSSize.data[0] != 2) {
      return;
    }

    VAddrIn_get();
    TieMemWrite(VAddrIn.data[0] & ~0x1, 2, &MemDataOut16.data[0]);
}


void
MemDataOut8_set(void)
{
    if (LSSize.data[0] != 1) {
      return;
    }

    VAddrIn_get();
    TieMemWrite(VAddrIn.data[0], 1, &MemDataOut8.data[0]);
}


void
Exception_set(void)
{
    /* Exception handling is not supported in native mode */
}


void
CPEnable_get(void)
{
    CPEnable.data[0] = 0xff; /* always enabled in native C mode */
}

#define RUR(n) ({ \
    int v; \
    switch (n) { \
    case 0: \
        v = RUR0(); break; \
    default: \
        fprintf(stderr, "Error: invalid rur number %d\n", n); \
        exit(-1); \
    } \
    v; \
})

#define WUR(v, n) \
    switch (n) { \
```

```
    case 0: \
        WUR0(v); break; \
    default: \
        fprintf(stderr, "Error: invalid wur number %d\n", n); \
        exit(-1); \
    }

gf8
GFADD8(gf8 gs_, gf8 gt_)
{
    /* operand variables */
    V1_t gr_o;
    V1_t gs_i;
    V1_t gt_i;
    /* unused operand variables */
    /* operand kill variables */
    V1_t gr_kill_o = {{0}};
    /* one-hot instruction signals */
    V1_t GFADD8 = {{1}};
    /* state variables */
    /* local wire variables */
    /* initialize in/inout operands */
    gs_i.data[0] = gs_;
    gt_i.data[0] = gt_;
    tie_load_instruction = 0;
    /* semantic statements */
    gr_o.data[0] = (gs_i.data[0] ^ gt_i.data[0]) & 0xff;
    gr_kill_o.data[0] = (0 & GFADD8.data[0]) & 0x1;
    /* write-back inout operands */
    /* return the output operand */
    return gr_o.data[0];
}

gf8
GFADD8I(gf8 gs_, int imm4_)
{
    /* operand variables */
    V1_t gr_o;
    V1_t gs_i;
    V1_t imm4;
    /* unused operand variables */
    /* operand kill variables */
    V1_t gr_kill_o = {{0}};
    /* one-hot instruction signals */
    V1_t GFADD8I = {{1}};
    /* state variables */
    /* local wire variables */
    /* initialize in/inout operands */
    gs_i.data[0] = gs_;
    imm4.data[0] = imm4_;
    tie_load_instruction = 0;
    /* semantic statements */
    gr_o.data[0] = (gs_i.data[0] ^ imm4.data[0]) & 0xff;
    gr_kill_o.data[0] = (0 & GFADD8I.data[0]) & 0x1;
    /* write-back inout operands */
    /* return the output operand */
    return gr_o.data[0];
```

```
}

gf8
GFMULX8(gf8 gs_)
{
    /* operand variables */
    V1_t gr_o;
    V1_t gs_i;
    /* unused operand variables */
    /* operand kill variables */
    V1_t gr_kill_o = {{0}};
    /* one-hot instruction signals */
    V1_t GFMULX8 = {{1}};
    /* state variables */
    V1_t gfmod_ps;
    /* local wire variables */
    V1_t tmp5;
    V1_t tmp4;
    V1_t tmp3;
    V1_t tmp2;
    V1_t tmp1;
    V1_t tmp0;
    /* get input state values */
    GetState(gfmod_ps, STATE_gfmod);
    /* initialize in/inout operands */
    gs_i.data[0] = gs_;
    tie_load_instruction = 0;
    /* semantic statements */
    tmp0.data[0] = (((gs_i.data[0] << 24) >> 31)) & 0x1;
    tmp1.data[0] = ((gs_i.data[0] & 0x7f)) & 0x7f;
    tmp2.data[0] = ((tmp1.data[0] << 1)|0) & 0xff;
    tmp3.data[0] = (tmp2.data[0] ^ gfmod_ps.data[0]) & 0xff;
    tmp4.data[0] = ((gs_i.data[0] & 0x7f)) & 0x7f;
    tmp5.data[0] = ((tmp4.data[0] << 1)|0) & 0xff;
    gr_o.data[0] = ((tmp0.data[0]) ? tmp3.data[0] : tmp5.data[0]) & 0xff;
    gr_kill_o.data[0] = (0 & GFMULX8.data[0]) & 0x1;
    /* write-back inout operands */
    /* return the output operand */
    return gr_o.data[0];
}

#define GFRWMOD8(gt) \
        GFRWMOD8_func(&(gt))

void
GFRWMOD8_func(gf8 *gt_)
{
    /* operand variables */
    V1_t gt_o;
    V1_t gt_i;
    /* unused operand variables */
    /* operand kill variables */
    V1_t gt_kill_o = {{0}};
    /* one-hot instruction signals */
    V1_t GFRWMOD8 = {{1}};
    /* state variables */
    V1_t gfmod_ps;
```

```
    V1_t gfmod_ns;
    V1_t gfmod_kill_ns;
    /* local wire variables */
    V1_t t1;
    V1_t t2;
    /* get input state values */
    GetState(gfmod_ps, STATE_gfmod);
    /* initialize in/inout operands */
    gt_i.data[0] = *gt_;
    tie_load_instruction = 0;
    /* semantic statements */
    t1.data[0] = gt_i.data[0] & 0xff;
    t2.data[0] = gfmod_ps.data[0] & 0xff;
    gfmod_ns.data[0] = t1.data[0] & 0xff;
    gt_o.data[0] = t2.data[0] & 0xff;
    gfmod_kill_ns.data[0] = (0 & GFRWMOD8.data[0]) & 0x1;
    gt_kill_o.data[0] = (0 & GFRWMOD8.data[0]) & 0x1;
    /* write-back inout operands */
    if (!gt_kill_o.data[0]) *gt_ = gt_o.data[0];
    /* update out/inout states */
    if (!gfmod_kill_ns.data[0]) SetState(STATE_gfmod, gfmod_ns);
}

gf8
LGF8_I(unsigned ars_, int imm8_)
{
    /* operand variables */
    V1_t gt_o;
    V1_t ars_i;
    V1_t imm8;
    /* unused operand variables */
    V1_t ars_o;
    V1_t gr_o;
    V1_t art_i = {{0}};
    /* operand kill variables */
    V1_t gt_kill_o = {{0}};
    V1_t ars_kill_o = {{0}};
    V1_t gr_kill_o = {{0}};
    /* one-hot instruction signals */
    V1_t LGF8_I = {{1}};
    V1_t LGF8_IU = {{0}};
    V1_t LGF8_X = {{0}};
    V1_t LGF8_XU = {{0}};
    /* state variables */
    /* local wire variables */
    V1_t tmp2;
    V1_t tmp1;
    V1_t tmp0;
    V1_t indexed;
    /* initialize in/inout operands */
    ars_i = *((V1_t *) &ars_);
    imm8.data[0] = imm8_;
    tie_load_instruction = 1;
    /* semantic statements */
    indexed.data[0] = (LGF8_X.data[0] | LGF8_XU.data[0]) & 0x1;
    LSSize.data[0] = 0x1 & 0x1f;
    VAddrBase.data[0] = ars_i.data[0];
```

```
    LSIndexed.data[0] = indexed.data[0] & 0x1;
    VAddrOffset.data[0] = imm8.data[0];
    VAddrIndex.data[0] = art_i.data[0];
    MemDataIn8_get();
    gt_o.data[0] = MemDataIn8.data[0] & 0xff;
    MemDataIn8_get();
    gr_o.data[0] = MemDataIn8.data[0] & 0xff;
    VAddrIn_get();
    ars_o.data[0] = VAddrIn.data[0];
    tmp0.data[0] = (LGF8_I.data[0] | LGF8_IU.data[0]) & 0x1;
    gt_kill_o.data[0] = (0 & tmp0.data[0]) & 0x1;
    tmp1.data[0] = (LGF8_IU.data[0] | LGF8_XU.data[0]) & 0x1;
    ars_kill_o.data[0] = (0 & tmp1.data[0]) & 0x1;
    tmp2.data[0] = (LGF8_X.data[0] | LGF8_XU.data[0]) & 0x1;
    gr_kill_o.data[0] = (0 & tmp2.data[0]) & 0x1;
    /* write-back inout operands */
    /* update output interface signals */
    /* return the output operand */
    return gt_o.data[0];
}

#define LGF8_IU(ars, imm8) \
        LGF8_IU_func(&(ars), imm8)

gf8
LGF8_IU_func(unsigned *ars_, int imm8_)
{
    /* operand variables */
    V1_t gt_o;
    V1_t ars_o;
    V1_t ars_i;
    V1_t imm8;
    /* unused operand variables */
    V1_t gr_o;
    V1_t art_i = {{0}};
    /* operand kill variables */
    V1_t gt_kill_o = {{0}};
    V1_t ars_kill_o = {{0}};
    V1_t gr_kill_o = {{0}};
    /* one-hot instruction signals */
    V1_t LGF8_I = {{0}};
    V1_t LGF8_IU = {{1}};
    V1_t LGF8_X = {{0}};
    V1_t LGF8_XU = {{0}};
    /* state variables */
    /* local wire variables */
    V1_t tmp2;
    V1_t tmp1;
    V1_t tmp0;
    V1_t indexed;
    /* initialize in/inout operands */
    ars_i = *((V1_t *) ars_);
    imm8.data[0] = imm8_;
    tie_load_instruction = 1;
    /* semantic statements */
    indexed.data[0] = (LGF8_X.data[0] | LGF8_XU.data[0]) & 0x1;
    LSSize.data[0] = 0x1 & 0x1f;
```

```
        VAddrBase.data[0] = ars_i.data[0];
        LSIndexed.data[0] = indexed.data[0] & 0x1;
        VAddrOffset.data[0] = imm8.data[0];
        VAddrIndex.data[0] = art_i.data[0];
        MemDataIn8_get();
        gt_o.data[0] = MemDataIn8.data[0] & 0xff;
        MemDataIn8_get();
        gr_o.data[0] = MemDataIn8.data[0] & 0xff;
        VAddrIn_get();
        ars_o.data[0] = VAddrIn.data[0];
        tmp0.data[0] = (LGF8_I.data[0] | LGF8_IU.data[0]) & 0x1;
        gt_kill_o.data[0] = (0 & tmp0.data[0]) & 0x1;
        tmp1.data[0] = (LGF8_IU.data[0] | LGF8_XU.data[0]) & 0x1;
        ars_kill_o.data[0] = (0 & tmp1.data[0]) & 0x1;
        tmp2.data[0] = (LGF8_X.data[0] | LGF8_XU.data[0]) & 0x1;
        gr_kill_o.data[0] = (0 & tmp2.data[0]) & 0x1;
        /* write-back inout operands */
        if (!ars_kill_o.data[0]) *ars_ = *((unsigned *) &ars_o);
        /* update output interface signals */
        /* return the output operand */
        return gt_o.data[0];
    }

    gf8
    LGF8_X(unsigned ars_, unsigned art_)
    {
        /* operand variables */
        V1_t gr_o;
        V1_t ars_i;
        V1_t art_i;
        /* unused operand variables */
        V1_t gt_o;
        V1_t ars_o;
        V1_t imm8 = {{0}};
        /* operand kill variables */
        V1_t gt_kill_o = {{0}};
        V1_t ars_kill_o = {{0}};
        V1_t gr_kill_o = {{0}};
        /* one-hot instruction signals */
        V1_t LGF8_I = {{0}};
        V1_t LGF8_IU = {{0}};
        V1_t LGF8_X = {{1}};
        V1_t LGF8_XU = {{0}};
        /* state variables */
        /* local wire variables */
        V1_t tmp2;
        V1_t tmp1;
        V1_t tmp0;
        V1_t indexed;
        /* initialize in/inout operands */
        ars_i = *((V1_t *) &ars_);
        art_i = *((V1_t *) &art_);
        tie_load_instruction = 1;
        /* semantic statements */
        indexed.data[0] = (LGF8_X.data[0] | LGF8_XU.data[0]) & 0x1;
        LSSize.data[0] = 0x1 & 0x1f;
        VAddrBase.data[0] = ars_i.data[0];
```

```
    LSIndexed.data[0] = indexed.data[0] & 0x1;
    VAddrOffset.data[0] = imm8.data[0];
    VAddrIndex.data[0] = art_i.data[0];
    MemDataIn8_get();
    gt_o.data[0] = MemDataIn8.data[0] & 0xff;
    MemDataIn8_get();
    gr_o.data[0] = MemDataIn8.data[0] & 0xff;
    VAddrIn_get();
    ars_o.data[0] = VAddrIn.data[0];
    tmp0.data[0] = (LGF8_I.data[0] | LGF8_IU.data[0]) & 0x1;
    gt_kill_o.data[0] = (0 & tmp0.data[0]) & 0x1;
    tmp1.data[0] = (LGF8_IU.data[0] | LGF8_XU.data[0]) & 0x1;
    ars_kill_o.data[0] = (0 & tmp1.data[0]) & 0x1;
    tmp2.data[0] = (LGF8_X.data[0] | LGF8_XU.data[0]) & 0x1;
    gr_kill_o.data[0] = (0 & tmp2.data[0]) & 0x1;
    /* write-back inout operands */
    /* update output interface signals */
    /* return the output operand */
    return gr_o.data[0];
}

#define LGF8_XU(ars, art) \
        LGF8_XU_func(&(ars), art)

gf8
LGF8_XU_func(unsigned *ars_, unsigned art_)
{
    /* operand variables */
    V1_t gr_o;
    V1_t ars_o;
    V1_t ars_i;
    V1_t art_i;
    /* unused operand variables */
    V1_t gt_o;
    V1_t imm8 = {{0}};
    /* operand kill variables */
    V1_t gt_kill_o = {{0}};
    V1_t ars_kill_o = {{0}};
    V1_t gr_kill_o = {{0}};
    /* one-hot instruction signals */
    V1_t LGF8_I = {{0}};
    V1_t LGF8_IU = {{0}};
    V1_t LGF8_X = {{0}};
    V1_t LGF8_XU = {{1}};
    /* state variables */
    /* local wire variables */
    V1_t tmp2;
    V1_t tmp1;
    V1_t tmp0;
    V1_t indexed;
    /* initialize in/inout operands */
    ars_i = *((V1_t *) ars_);
    art_i = *((V1_t *) &art_);
    tie_load_instruction = 1;
    /* semantic statements */
    indexed.data[0] = (LGF8_X.data[0] | LGF8_XU.data[0]) & 0x1;
    LSSize.data[0] = 0x1 & 0x1f;
```

```
    VAddrBase.data[0] = ars_i.data[0];
    LSIndexed.data[0] = indexed.data[0] & 0x1;
    VAddrOffset.data[0] = imm8.data[0];
    VAddrIndex.data[0] = art_i.data[0];
    MemDataIn8_get();
    gt_o.data[0] = MemDataIn8.data[0] & 0xff;
    MemDataIn8_get();
    gr_o.data[0] = MemDataIn8.data[0] & 0xff;
    VAddrIn_get();
    ars_o.data[0] = VAddrIn.data[0];
    tmp0.data[0] = (LGF8_I.data[0] | LGF8_IU.data[0]) & 0x1;
    gt_kill_o.data[0] = (0 & tmp0.data[0]) & 0x1;
    tmp1.data[0] = (LGF8_IU.data[0] | LGF8_XU.data[0]) & 0x1;
    ars_kill_o.data[0] = (0 & tmp1.data[0]) & 0x1;
    tmp2.data[0] = (LGF8_X.data[0] | LGF8_XU.data[0]) & 0x1;
    gr_kill_o.data[0] = (0 & tmp2.data[0]) & 0x1;
    /* write-back inout operands */
    if (!ars_kill_o.data[0]) *ars_ = *((unsigned *) &ars_o);
    /* update output interface signals */
    /* return the output operand */
    return gr_o.data[0];
}

void
SGF8_I(gf8 gt_, unsigned ars_, int imm8_)
{
    /* operand variables */
    V1_t gt_i;
    V1_t ars_i;
    V1_t imm8;
    /* unused operand variables */
    V1_t ars_o;
    V1_t gr_i = {{0}};
    V1_t art_i = {{0}};
    /* operand kill variables */
    V1_t ars_kill_o = {{0}};
    /* one-hot instruction signals */
    V1_t SGF8_IU = {{0}};
    V1_t SGF8_X = {{0}};
    V1_t SGF8_XU = {{0}};
    /* state variables */
    /* local wire variables */
    V1_t tmp1;
    V1_t tmp0;
    V1_t indexed;
    /* initialize in/inout operands */
    gt_i.data[0] = gt_;
    ars_i = *((V1_t *) &ars_);
    imm8.data[0] = imm8_;
    tie_load_instruction = 0;
    /* semantic statements */
    indexed.data[0] = (SGF8_X.data[0] | SGF8_XU.data[0]) & 0x1;
    LSSize.data[0] = 0x1 & 0x1f;
    VAddrBase.data[0] = ars_i.data[0];
    LSIndexed.data[0] = indexed.data[0] & 0x1;
    VAddrOffset.data[0] = imm8.data[0];
    VAddrIndex.data[0] = art_i.data[0];
```

```
    tmp0.data[0] = (SGF8_X.data[0] | SGF8_XU.data[0]) & 0x1;
    MemDataOut8.data[0] = ((tmp0.data[0]) ? gr_i.data[0] : gt_i.data[0]) &
0xff;
    VAddrIn_get();
    ars_o.data[0] = VAddrIn.data[0];
    tmp1.data[0] = (SGF8_IU.data[0] | SGF8_XU.data[0]) & 0x1;
    ars_kill_o.data[0] = (0 & tmp1.data[0]) & 0x1;
    /* write-back inout operands */
    /* update output interface signals */
    MemDataOut8_set();
}

#define SGF8_IU(gt, ars, imm8) \
        SGF8_IU_func(gt, &(ars), imm8)

void
SGF8_IU_func(gf8 gt_, unsigned *ars_, int imm8_)
{
    /* operand variables */
    V1_t gt_i;
    V1_t ars_o;
    V1_t ars_i;
    V1_t imm8;
    /* unused operand variables */
    V1_t gr_i = {{0}};
    V1_t art_i = {{0}};
    /* operand kill variables */
    V1_t ars_kill_o = {{0}};
    /* one-hot instruction signals */
    V1_t SGF8_IU = {{1}};
    V1_t SGF8_X = {{0}};
    V1_t SGF8_XU = {{0}};
    /* state variables */
    /* local wire variables */
    V1_t tmp1;
    V1_t tmp0;
    V1_t indexed;
    /* initialize in/inout operands */
    gt_i.data[0] = gt_;
    ars_i = *((V1_t *) ars_);
    imm8.data[0] = imm8_;
    tie_load_instruction = 0;
    /* semantic statements */
    indexed.data[0] = (SGF8_X.data[0] | SGF8_XU.data[0]) & 0x1;
    LSSize.data[0] = 0x1 & 0x1f;
    VAddrBase.data[0] = ars_i.data[0];
    LSIndexed.data[0] = indexed.data[0] & 0x1;
    VAddrOffset.data[0] = imm8.data[0];
    VAddrIndex.data[0] = art_i.data[0];
    tmp0.data[0] = (SGF8_X.data[0] | SGF8_XU.data[0]) & 0x1;
    MemDataOut8.data[0] = ((tmp0.data[0]) ? gr_i.data[0] : gt_i.data[0]) &
0xff;
    VAddrIn_get();
    ars_o.data[0] = VAddrIn.data[0];
    tmp1.data[0] = (SGF8_IU.data[0] | SGF8_XU.data[0]) & 0x1;
    ars_kill_o.data[0] = (0 & tmp1.data[0]) & 0x1;
    /* write-back inout operands */
```

```
    if (!ars_kill_o.data[0]) *ars_ = *((unsigned *) &ars_o);
    /* update output interface signals */
    MemDataOut8_set();
}

void
SGF8_X(gf8 gr_, unsigned ars_, unsigned art_)
{
    /* operand variables */
    V1_t gr_i;
    V1_t ars_i;
    V1_t art_i;
    /* unused operand variables */
    V1_t gt_i = {{0}};
    V1_t ars_o;
    V1_t imm8 = {{0}};
    /* operand kill variables */
    V1_t ars_kill_o = {{0}};
    /* one-hot instruction signals */
    V1_t SGF8_IU = {{0}};
    V1_t SGF8_X = {{1}};
    V1_t SGF8_XU = {{0}};
    /* state variables */
    /* local wire variables */
    V1_t tmp1;
    V1_t tmp0;
    V1_t indexed;
    /* initialize in/inout operands */
    gr_i.data[0] = gr_;
    ars_i = *((V1_t *) &ars_);
    art_i = *((V1_t *) &art_);
    tie_load_instruction = 0;
    /* semantic statements */
    indexed.data[0] = (SGF8_X.data[0] | SGF8_XU.data[0]) & 0x1;
    LSSize.data[0] = 0x1 & 0x1f;
    VAddrBase.data[0] = ars_i.data[0];
    LSIndexed.data[0] = indexed.data[0] & 0x1;
    VAddrOffset.data[0] = imm8.data[0];
    VAddrIndex.data[0] = art_i.data[0];
    tmp0.data[0] = (SGF8_X.data[0] | SGF8_XU.data[0]) & 0x1;
    MemDataOut8.data[0] = ((tmp0.data[0]) ? gr_i.data[0] : gt_i.data[0]) &
0xff;
    VAddrIn_get();
    ars_o.data[0] = VAddrIn.data[0];
    tmp1.data[0] = (SGF8_IU.data[0] | SGF8_XU.data[0]) & 0x1;
    ars_kill_o.data[0] = (0 & tmp1.data[0]) & 0x1;
    /* write-back inout operands */
    /* update output interface signals */
    MemDataOut8_set();
}

#define SGF8_XU(gr, ars, art) \
        SGF8_XU_func(gr, &(ars), art)

void
SGF8_XU_func(gf8 gr_, unsigned *ars_, unsigned art_)
{
```

```
    /* operand variables */
    V1_t gr_i;
    V1_t ars_o;
    V1_t ars_i;
    V1_t art_i;
    /* unused operand variables */
    V1_t gt_i = {{0}};
    V1_t imm8 = {{0}};
    /* operand kill variables */
    V1_t ars_kill_o = {{0}};
    /* one-hot instruction signals */
    V1_t SGF8_IU = {{0}};
    V1_t SGF8_X = {{0}};
    V1_t SGF8_XU = {{1}};
    /* state variables */
    /* local wire variables */
    V1_t tmp1;
    V1_t tmp0;
    V1_t indexed;
    /* initialize in/inout operands */
    gr_i.data[0] = gr_;
    ars_i = *((V1_t *) ars_);
    art_i = *((V1_t *) &art_);
    tie_load_instruction = 0;
    /* semantic statements */
    indexed.data[0] = (SGF8_X.data[0] | SGF8_XU.data[0]) & 0x1;
    LSSize.data[0] = 0x1 & 0x1f;
    VAddrBase.data[0] = ars_i.data[0];
    LSIndexed.data[0] = indexed.data[0] & 0x1;
    VAddrOffset.data[0] = imm8.data[0];
    VAddrIndex.data[0] = art_i.data[0];
    tmp0.data[0] = (SGF8_X.data[0] | SGF8_XU.data[0]) & 0x1;
    MemDataOut8.data[0] = ((tmp0.data[0]) ? gr_i.data[0] : gt_i.data[0]) &
0xff;
    VAddrIn_get();
    ars_o.data[0] = VAddrIn.data[0];
    tmp1.data[0] = (SGF8_IU.data[0] | SGF8_XU.data[0]) & 0x1;
    ars_kill_o.data[0] = (0 & tmp1.data[0]) & 0x1;
    /* write-back inout operands */
    if (!ars_kill_o.data[0]) *ars_ = *((unsigned *) &ars_o);
    /* update output interface signals */
    MemDataOut8_set();
}

unsigned
RUR0()
{
    /* operand variables */
    V1_t arr_o;
    /* unused operand variables */
    /* operand kill variables */
    V1_t arr_kill_o = {{0}};
    /* one-hot instruction signals */
    V1_t RUR0 = {{1}};
    /* state variables */
    V1_t gfmod_ps;
    /* local wire variables */
```

```
    /* get input state values */
    GetState(gfmod_ps, STATE_gfmod);
    /* initialize in/inout operands */
    tie_load_instruction = 0;
    /* semantic statements */
    arr_o.data[0] = gfmod_ps.data[0];
    arr_kill_o.data[0] = (0 & RUR0.data[0]) & 0x1;
    /* write-back inout operands */
    /* return the output operand */
    return *((unsigned *) &arr_o);
}

void
WUR0(unsigned art_)
{
    /* operand variables */
    V1_t art_i;
    /* unused operand variables */
    /* operand kill variables */
    /* one-hot instruction signals */
    V1_t WUR0 = {{1}};
    /* state variables */
    V1_t gfmod_ns;
    V1_t gfmod_kill_ns;
    /* local wire variables */
    V1_t tmp0;
    /* initialize in/inout operands */
    art_i = *((V1_t *) &art_);
    tie_load_instruction = 0;
    /* semantic statements */
    tmp0.data[0] = ((art_i.data[0] & 0xff)) & 0xff;
    gfmod_ns.data[0] = (tmp0.data[0]) & 0xff;
    gfmod_kill_ns.data[0] = (0 & WUR0.data[0]) & 0x1;
    /* write-back inout operands */
    /* update out/inout states */
    if (!gfmod_kill_ns.data[0]) SetState(STATE_gfmod, gfmod_ns);
}

#define gf8_loadi(_s, o) ({ \
    gf8 t; \
    gf8 *s = _s; \
    gf8 LGF8_I_return; \
    LGF8_I_return = LGF8_I(*((unsigned *)&(s)), *((int *)&(o))); \
    t = *((gf8 *)&LGF8_I_return); \
    t; \
})

#define gf8_storei(_t, _s, o) ({ \
    gf8 t = _t; \
    gf8 *s = _s; \
    SGF8_I(*((gf8 *)&(t)), *((unsigned *)&(s)), *((int *)&(o))); \
})

#define gf8_move(_r, _s) ({ \
    gf8 r = _r; \
    gf8 s = _s; \
    gf8 GFADD8_return; \
```

```
    GFADD8_return = GFADD8(*((gf8 *)&(s)), *((gf8 *)&(0))); \
    r = *((gf8 *)&GFADD8_return); \
})

#ifdef TIE_DEBUG
#undef gf8_loadi
#undef gf8_storei
#undef gf8_move
#undef GFADD8
#undef GFADD8I
#undef GFMULX8
#undef GFRWMOD8
#undef LGF8_I
#undef SGF8_I
#undef LGF8_IU
#undef SGF8_IU
#undef LGF8_X
#undef SGF8_X
#undef LGF8_XU
#undef SGF8_XU
#undef RUR0
#undef WUR0
#endif
#endif
```

cstub-gf-ref.c

```
#ifndef __XTENSA__
#ifdef TIE_DEBUG
#define gf8_loadi TIE_gf8_loadi
#define gf8_storei TIE_gf8_storei
#define gf8_move TIE_gf8_move
#define GFADD8 TIE_GFADD8
#define GFADD8I TIE_GFADD8I
#define GFMULX8 TIE_GFMULX8
#define GFRWMOD8 TIE_GFRWMOD8
#define LGF8_I TIE_LGF8_I
#define SGF8_I TIE_SGF8_I
#define LGF8_IU TIE_LGF8_IU
#define SGF8_IU TIE_SGF8_IU
#define LGF8_X TIE_LGF8_X
#define SGF8_X TIE_SGF8_X
#define LGF8_XU TIE_LGF8_XU
#define SGF8_XU TIE_SGF8_XU
#define RUR0 TIE_RUR0
#define WUR0 TIE_WUR0
#endif

#include <stdio.h>
#define LittleEndian    0
#define BigEndian 1
#define PIFReadDataBits 128
#define PIFWriteDataBits      128
#define IsaMemoryOrder  LittleEndian
#include "BR.h"
#include "LS.h"
```

```
#define BPW 32
#define WINDEX(_n) ((_n) / BPW)
#define BINDEX(_n) ((_n) % BPW)

typedef unsigned char Vb_t;
typedef unsigned short Vs_t;
typedef struct V1_s {unsigned data[1];} V1_t;
typedef struct V2_s {unsigned data[2];} V2_t;
typedef struct V4_s {unsigned data[4];} V4_t;

typedef Vb_t gf8;



static int tie_load_instruction = 0;

void
TieMemRead(unsigned *data, unsigned addr)
{
    unsigned char *mem;
    unsigned modulus, bytes, offset;
    int t, b0, b1, b2, b3;

    bytes = PIFReadDataBits / 8;
    modulus = bytes - 1;
    mem = (unsigned char *) (addr & ~modulus);
    offset = (unsigned char *) addr - mem;
    if (IsaMemoryOrder == LittleEndian) {
      for(t = 0; t < bytes/sizeof(int); t++) {
          b0 = mem[(offset++) & modulus];
          b1 = mem[(offset++) & modulus];
          b2 = mem[(offset++) & modulus];
          b3 = mem[(offset++) & modulus];
          data[t] = (b3 << 24) | (b2 << 16) | (b1 << 8) | b0;
      }
    } else {
      for(t = bytes/sizeof(int) - 1; t >= 0; t--) {
          b3 = mem[(offset++) & modulus];
          b2 = mem[(offset++) & modulus];
          b1 = mem[(offset++) & modulus];
          b0 = mem[(offset++) & modulus];
          data[t] = (b3 << 24) | (b2 << 16) | (b1 << 8) | b0;
      }
    }
}


void
TieMemWrite(unsigned addr, unsigned bytes, unsigned *data)
{
    unsigned char *mem;
    unsigned modulus, offset, w;
    int t;

    if (PIFWriteDataBits < bytes * 8) {
        fprintf(stderr, "Error: not configured to write %d bytes\n", bytes);
        exit(1);
```

```
    }

    modulus = bytes - 1;
    mem = (unsigned char *) (addr & ~modulus);
    if (IsaMemoryOrder == LittleEndian) {
      if (bytes == 1) {
          mem[0] = data[0] & 0xff;
      } else if (bytes == 2) {
          mem[0] = data[0] & 0xff;
          mem[1] = (data[0] >> 8) & 0xff;
      } else {
          offset = 0;
          for(t = 0; t < bytes/sizeof(int); t++) {
            w = data[t];
            mem[offset++] = w & 255;
            mem[offset++] = (w >> 8) & 255;
            mem[offset++] = (w >> 16) & 255;
            mem[offset++] = (w >> 24) & 255;
          }
      }
    } else {
      if (bytes == 1) {
          mem[0] = data[0] & 0xff;
      } else if (bytes == 2) {
          mem[1] = data[0] & 0xff;
          mem[0] = (data[0] >> 8) & 0xff;
      } else {
          offset = 0;
          for(t = bytes/sizeof(int) - 1; t >= 0; t--) {
            w = data[t];
            mem[offset++] = (w >> 24) & 255;
            mem[offset++] = (w >> 16) & 255;
            mem[offset++] = (w >> 8) & 255;
            mem[offset++] = w & 255;
          }
      }
    }
}
#define GetState(_s, _n) _s = _n
#define SetState(_n, _s) _n = _s

V1_t STATE_gfmod;

V1_t VAddr = {{0}};
V1_t VAddrBase = {{0}};
V1_t VAddrOffset = {{0}};
V1_t VAddrIndex = {{0}};
V1_t VAddrIn = {{0}};
V1_t LSSize = {{0}};
V1_t LSIndexed = {{0}};
V4_t MemDataIn128 = {{0,0,0,0}};
V2_t MemDataIn64 = {{0,0}};
V1_t MemDataIn32 = {{0}};
V1_t MemDataIn16 = {{0}};
V1_t MemDataIn8 = {{0}};
V4_t MemDataOut128 = {{0,0,0,0}};
V2_t MemDataOut64 = {{0,0}};
```

```
V1_t MemDataOut32 = {{0}};
V1_t MemDataOut16 = {{0}};
V1_t MemDataOut8 = {{0}};
V1_t Exception = {{0}};
V1_t ExcCause = {{0}};
V1_t CPEnable = {{0}};


void
VAddrIn_get(void)
{
    if (LSIndexed.data[0] != 0) {
        VAddrIn.data[0] = VAddrBase.data[0] + VAddrIndex.data[0];
    } else {
        VAddrIn.data[0] = VAddrBase.data[0] + VAddrOffset.data[0];
    }
}


void
MemDataIn128_get(void)
{
    unsigned data[4];

    if ((!tie_load_instruction) || (LSSize.data[0] != 16)) {
      return;
    }

    if (PIFReadDataBits < 128) {
      fprintf(stderr, "Error: not configured to read 16 bytes\n");
      exit(-1);
    }

    VAddrIn_get();
    TieMemRead(&data[0], VAddrIn.data[0]);
    MemDataIn128.data[0] = data[0];
    MemDataIn128.data[1] = data[1];
    MemDataIn128.data[2] = data[2];
    MemDataIn128.data[3] = data[3];
}


void
MemDataIn64_get(void)
{
    unsigned data[4];

    if ((!tie_load_instruction) || (LSSize.data[0] != 8)) {
      return;
    }

    if (PIFReadDataBits < 64) {
      fprintf(stderr, "Error: not configured to read 8 bytes\n");
      exit(-1);
    }

    VAddrIn_get();
```

```
    TieMemRead(&data[0], VAddrIn.data[0]);
    if (IsaMemoryOrder == LittleEndian) {
      MemDataIn64.data[0] = data[0];
      MemDataIn64.data[1] = data[1];
    } else if (PIFReadDataBits == 64) {
      MemDataIn64.data[0] = data[0];
      MemDataIn64.data[1] = data[1];
    } else {
      MemDataIn64.data[0] = data[2];
      MemDataIn64.data[1] = data[3];
    }
}


void
MemDataIn32_get(void)
{
    unsigned data[4];

    if ((!tie_load_instruction) || (LSSize.data[0] != 4)) {
      return;
    }

    if (PIFReadDataBits < 32) {
      fprintf(stderr, "Error: not configured to read 4 bytes\n");
      exit(-1);
    }

    VAddrIn_get();
    TieMemRead(&data[0], VAddrIn.data[0]);
    if (IsaMemoryOrder == LittleEndian) {
      MemDataIn32.data[0] = data[0];
    } else if (PIFReadDataBits == 32) {
      MemDataIn32.data[0] = data[0];
    } else if (PIFReadDataBits == 64) {
      MemDataIn32.data[0] = data[1];
    } else {
      MemDataIn32.data[0] = data[3];
    }
}


void
MemDataIn16_get(void)
{
    unsigned data[4];

    if ((!tie_load_instruction) || (LSSize.data[0] != 2)) {
      return;
    }

    if (PIFReadDataBits < 16) {
      fprintf(stderr, "Error: not configured to read 2 bytes\n");
      exit(-1);
    }

    VAddrIn_get();
```

```
    TieMemRead(&data[0], VAddrIn.data[0]);
    if (IsaMemoryOrder == LittleEndian) {
      MemDataIn16.data[0] = data[0] & 0xffff;
    } else if (PIFReadDataBits == 32) {
      MemDataIn16.data[0] = data[0] >> 16;
    } else if (PIFReadDataBits == 64) {
      MemDataIn16.data[0] = data[1] >> 16;
    } else {
      MemDataIn16.data[0] = data[3] >> 16;
    }
}


void
MemDataIn8_get(void)
{
    unsigned data[4];

    if ((!tie_load_instruction) || (LSSize.data[0] != 1)) {
      return;
    }

    if (PIFReadDataBits < 8) {
      fprintf(stderr, "Error: not configured to read 1 byte\n");
      exit(-1);
    }

    VAddrIn_get();
    TieMemRead(&data[0], VAddrIn.data[0]);
    if (IsaMemoryOrder == LittleEndian) {
      MemDataIn8.data[0] = data[0] & 0xff;
    } else if (PIFReadDataBits == 32) {
      MemDataIn8.data[0] = data[0] >> 24;
    } else if (PIFReadDataBits == 64) {
      MemDataIn8.data[0] = data[1] >> 24;
    } else {
      MemDataIn8.data[0] = data[3] >> 24;
    }
}


void
MemDataOut128_set(void)
{
    if (LSSize.data[0] != 16) {
      return;
    }

    VAddrIn_get();
    TieMemWrite(VAddrIn.data[0] & ~0xf, 16, &MemDataOut128.data[0]);
}


void
MemDataOut64_set(void)
{
    if (LSSize.data[0] != 8) {
```

```
        return;
    }

    VAddrIn_get();
    TieMemWrite(VAddrIn.data[0] & ~0x7, 8, &MemDataOut64.data[0]);
}


void
MemDataOut32_set(void)
{
    if (LSSize.data[0] != 4) {
        return;
    }

    VAddrIn_get();
    TieMemWrite(VAddrIn.data[0] & ~0x3, 4, &MemDataOut32.data[0]);
}


void
MemDataOut16_set(void)
{
    if (LSSize.data[0] != 2) {
        return;
    }

    VAddrIn_get();
    TieMemWrite(VAddrIn.data[0] & ~0x1, 2, &MemDataOut16.data[0]);
}


void
MemDataOut8_set(void)
{
    if (LSSize.data[0] != 1) {
        return;
    }

    VAddrIn_get();
    TieMemWrite(VAddrIn.data[0], 1, &MemDataOut8.data[0]);
}


void
Exception_set(void)
{
    /* Exception handling is not supported in native mode */
}


void
CPEnable_get(void)
{
    CPEnable.data[0] = 0xff; /* always enabled in native C mode */
}
```

```
#define RUR(n) ({ \
    int v; \
    switch (n) { \
    case 0: \
        v = RUR0(); break; \
    default: \
        fprintf(stderr, "Error: invalid rur number %d\n", n); \
        exit(-1); \
    } \
    v; \
})

#define WUR(v, n) \
    switch (n) { \
    case 0: \
        WUR0(v); break; \
    default: \
        fprintf(stderr, "Error: invalid wur number %d\n", n); \
        exit(-1); \
    }

gf8
GFADD8(gf8 gs_, gf8 gt_)
{
    /* operand variables */
    V1_t gr_o;
    V1_t gs_i;
    V1_t gt_i;
    /* unused operand variables */
    /* operand kill variables */
    V1_t gr_kill_o = {{0}};
    /* one-hot instruction signals */
    V1_t GFADD8 = {{1}};
    /* state variables */
    /* local wire variables */
    /* initialize in/inout operands */
    gs_i.data[0] = gs_;
    gt_i.data[0] = gt_;
    tie_load_instruction = 0;
    /* semantic statements */
    gr_o.data[0] = (gs_i.data[0] ^ gt_i.data[0]) & 0xff;
    gr_kill_o.data[0] = (0 & GFADD8.data[0]) & 0x1;
    /* write-back inout operands */
    /* return the output operand */
    return gr_o.data[0];
}

gf8
GFADD8I(gf8 gs_, int imm4_)
{
    /* operand variables */
    V1_t gr_o;
    V1_t gs_i;
    V1_t imm4;
    /* unused operand variables */
    /* operand kill variables */
    V1_t gr_kill_o = {{0}};
```

```
    /* one-hot instruction signals */
    V1_t GFADD8I = {{1}};
    /* state variables */
    /* local wire variables */
    /* initialize in/inout operands */
    gs_i.data[0] = gs_;
    imm4.data[0] = imm4_;
    tie_load_instruction = 0;
    /* semantic statements */
    gr_o.data[0] = (gs_i.data[0] ^ imm4.data[0]) & 0xff;
    gr_kill_o.data[0] = (0 & GFADD8I.data[0]) & 0x1;
    /* write-back inout operands */
    /* return the output operand */
    return gr_o.data[0];
}

gf8
GFMULX8(gf8 gs_)
{
    /* operand variables */
    V1_t gr_o;
    V1_t gs_i;
    /* unused operand variables */
    /* operand kill variables */
    V1_t gr_kill_o = {{0}};
    /* one-hot instruction signals */
    V1_t GFMULX8 = {{1}};
    /* state variables */
    V1_t gfmod_ps;
    /* local wire variables */
    V1_t tmp5;
    V1_t tmp4;
    V1_t tmp3;
    V1_t tmp2;
    V1_t tmp1;
    V1_t tmp0;
    /* get input state values */
    GetState(gfmod_ps, STATE_gfmod);
    /* initialize in/inout operands */
    gs_i.data[0] = gs_;
    tie_load_instruction = 0;
    /* semantic statements */
    tmp0.data[0] = (((gs_i.data[0] << 24) >> 31)) & 0x1;
    tmp1.data[0] = ((gs_i.data[0] & 0x7f)) & 0x7f;
    tmp2.data[0] = ((tmp1.data[0] << 1)|0) & 0xff;
    tmp3.data[0] = (tmp2.data[0] ^ gfmod_ps.data[0]) & 0xff;
    tmp4.data[0] = ((gs_i.data[0] & 0x7f)) & 0x7f;
    tmp5.data[0] = ((tmp4.data[0] << 1)|0) & 0xff;
    gr_o.data[0] = ((tmp0.data[0]) ? tmp3.data[0] : tmp5.data[0]) & 0xff;
    gr_kill_o.data[0] = (0 & GFMULX8.data[0]) & 0x1;
    /* write-back inout operands */
    /* return the output operand */
    return gr_o.data[0];
}

#define GFRWMOD8(gt) \
        GFRWMOD8_func(&(gt))
```

```
void
GFRWMOD8_func(gf8 *gt_)
{
    /* operand variables */
    V1_t gt_o;
    V1_t gt_i;
    /* unused operand variables */
    /* operand kill variables */
    V1_t gt_kill_o = {{0}};
    /* one-hot instruction signals */
    V1_t GFRWMOD8 = {{1}};
    /* state variables */
    V1_t gfmod_ps;
    V1_t gfmod_ns;
    V1_t gfmod_kill_ns;
    /* local wire variables */
    V1_t t1;
    V1_t t2;
    /* get input state values */
    GetState(gfmod_ps, STATE_gfmod);
    /* initialize in/inout operands */
    gt_i.data[0] = *gt_;
    tie_load_instruction = 0;
    /* semantic statements */
    t1.data[0] = gt_i.data[0] & 0xff;
    t2.data[0] = gfmod_ps.data[0] & 0xff;
    gfmod_ns.data[0] = t1.data[0] & 0xff;
    gt_o.data[0] = t2.data[0] & 0xff;
    gfmod_kill_ns.data[0] = (0 & GFRWMOD8.data[0]) & 0x1;
    gt_kill_o.data[0] = (0 & GFRWMOD8.data[0]) & 0x1;
    /* write-back inout operands */
    if (!gt_kill_o.data[0]) *gt_ = gt_o.data[0];
    /* update out/inout states */
    if (!gfmod_kill_ns.data[0]) SetState(STATE_gfmod, gfmod_ns);
}

gf8
LGF8_I(unsigned ars_, int imm8_)
{
    /* operand variables */
    V1_t gt_o;
    V1_t ars_i;
    V1_t imm8;
    /* unused operand variables */
    /* operand kill variables */
    V1_t gt_kill_o = {{0}};
    /* one-hot instruction signals */
    V1_t LGF8_I = {{1}};
    /* state variables */
    /* local wire variables */
    /* initialize in/inout operands */
    ars_i = *((V1_t *) &ars_);
    imm8.data[0] = imm8_;
    tie_load_instruction = 1;
    /* semantic statements */
    LSSize.data[0] = 0x1 & 0x1f;
```

```
    VAddrBase.data[0] = ars_i.data[0];
    LSIndexed.data[0] = 0 & 0x1;
    VAddrOffset.data[0] = imm8.data[0];
    MemDataIn8_get();
    gt_o.data[0] = MemDataIn8.data[0] & 0xff;
    gt_kill_o.data[0] = (0 & LGF8_I.data[0]) & 0x1;
    /* write-back inout operands */
    /* update output interface signals */
    /* return the output operand */
    return gt_o.data[0];
}

#define LGF8_IU(ars, imm8) \
        LGF8_IU_func(&(ars), imm8)

gf8
LGF8_IU_func(unsigned *ars_, int imm8_)
{
    /* operand variables */
    V1_t gt_o;
    V1_t ars_o;
    V1_t ars_i;
    V1_t imm8;
    /* unused operand variables */
    /* operand kill variables */
    V1_t gt_kill_o = {{0}};
    V1_t ars_kill_o = {{0}};
    /* one-hot instruction signals */
    V1_t LGF8_IU = {{1}};
    /* state variables */
    /* local wire variables */
    /* initialize in/inout operands */
    ars_i = *((V1_t *) ars_);
    imm8.data[0] = imm8_;
    tie_load_instruction = 1;
    /* semantic statements */
    LSSize.data[0] = 0x1 & 0x1f;
    VAddrBase.data[0] = ars_i.data[0];
    LSIndexed.data[0] = 0 & 0x1;
    VAddrOffset.data[0] = imm8.data[0];
    MemDataIn8_get();
    gt_o.data[0] = MemDataIn8.data[0] & 0xff;
    VAddrIn_get();
    ars_o.data[0] = VAddrIn.data[0];
    gt_kill_o.data[0] = (0 & LGF8_IU.data[0]) & 0x1;
    ars_kill_o.data[0] = (0 & LGF8_IU.data[0]) & 0x1;
    /* write-back inout operands */
    if (!ars_kill_o.data[0]) *ars_ = *((unsigned *) &ars_o);
    /* update output interface signals */
    /* return the output operand */
    return gt_o.data[0];
}

gf8
LGF8_X(unsigned ars_, unsigned art_)
{
    /* operand variables */
```

```
    V1_t gr_o;
    V1_t ars_i;
    V1_t art_i;
    /* unused operand variables */
    /* operand kill variables */
    V1_t gr_kill_o = {{0}};
    /* one-hot instruction signals */
    V1_t LGF8_X = {{1}};
    /* state variables */
    /* local wire variables */
    /* initialize in/inout operands */
    ars_i = *((V1_t *) &ars_);
    art_i = *((V1_t *) &art_);
    tie_load_instruction = 1;
    /* semantic statements */
    LSSize.data[0] = 0x1 & 0x1f;
    VAddrBase.data[0] = ars_i.data[0];
    LSIndexed.data[0] = 0x1 & 0x1;
    VAddrIndex.data[0] = art_i.data[0];
    MemDataIn8_get();
    gr_o.data[0] = MemDataIn8.data[0] & 0xff;
    VAddrIn_get();
    ars_o.data[0] = VAddrIn.data[0];
    gr_kill_o.data[0] = (0 & LGF8_X.data[0]) & 0x1;
    /* write-back inout operands */
    /* update output interface signals */
    /* return the output operand */
    return gr_o.data[0];
}

#define LGF8_XU(ars, art) \
        LGF8_XU_func(&(ars), art)

gf8
LGF8_XU_func(unsigned *ars_, unsigned art_)
{
    /* operand variables */
    V1_t gr_o;
    V1_t ars_o;
    V1_t ars_i;
    V1_t art_i;
    /* unused operand variables */
    /* operand kill variables */
    V1_t gr_kill_o = {{0}};
    V1_t ars_kill_o = {{0}};
    /* one-hot instruction signals */
    V1_t LGF8_XU = {{1}};
    /* state variables */
    /* local wire variables */
    /* initialize in/inout operands */
    ars_i = *((V1_t *) ars_);
    art_i = *((V1_t *) &art_);
    tie_load_instruction = 1;
    /* semantic statements */
    LSSize.data[0] = 0x1 & 0x1f;
    VAddrBase.data[0] = ars_i.data[0];
    LSIndexed.data[0] = 0x1 & 0x1;
```

```
    VAddrIndex.data[0] = art_i.data[0];
    MemDataIn8_get();
    gr_o.data[0] = MemDataIn8.data[0] & 0xff;
    VAddrIn_get();
    ars_o.data[0] = VAddrIn.data[0];
    gr_kill_o.data[0] = (0 & LGF8_XU.data[0]) & 0x1;
    ars_kill_o.data[0] = (0 & LGF8_XU.data[0]) & 0x1;
    /* write-back inout operands */
    if (!ars_kill_o.data[0]) *ars_ = *((unsigned *) &ars_o);
    /* update output interface signals */
    /* return the output operand */
    return gr_o.data[0];
}

void
SGF8_I(gf8 gt_, unsigned ars_, int imm8_)
{
    /* operand variables */
    V1_t gt_i;
    V1_t ars_i;
    V1_t imm8;
    /* unused operand variables */
    /* operand kill variables */
    /* one-hot instruction signals */
    /* state variables */
    /* local wire variables */
    /* initialize in/inout operands */
    gt_i.data[0] = gt_;
    ars_i = *((V1_t *) &ars_);
    imm8.data[0] = imm8_;
    tie_load_instruction = 0;
    /* semantic statements */
    LSSize.data[0] = 0x1 & 0x1f;
    VAddrBase.data[0] = ars_i.data[0];
    LSIndexed.data[0] = 0 & 0x1;
    VAddrOffset.data[0] = imm8.data[0];
    MemDataOut8.data[0] = gt_i.data[0] & 0xff;
    /* write-back inout operands */
    /* update output interface signals */
    MemDataOut8_set();
}

#define SGF8_IU(gt, ars, imm8) \
        SGF8_IU_func(gt, &(ars), imm8)

void
SGF8_IU_func(gf8 gt_, unsigned *ars_, int imm8_)
{
    /* operand variables */
    V1_t gt_i;
    V1_t ars_o;
    V1_t ars_i;
    V1_t imm8;
    /* unused operand variables */
    /* operand kill variables */
    V1_t ars_kill_o = {{0}};
    /* one-hot instruction signals */
```

```
    V1_t SGF8_IU = {{1}};
    /* state variables */
    /* local wire variables */
    /* initialize in/inout operands */
    gt_i.data[0] = gt_;
    ars_i = *((V1_t *) ars_);
    imm8.data[0] = imm8_;
    tie_load_instruction = 0;
    /* semantic statements */
    LSSize.data[0] = 0x1 & 0x1f;
    VAddrBase.data[0] = ars_i.data[0];
    LSIndexed.data[0] = 0 & 0x1;
    VAddrOffset.data[0] = imm8.data[0];
    MemDataOut8.data[0] = gt_i.data[0] & 0xff;
    VAddrIn_get();
    ars_o.data[0] = VAddrIn.data[0];
    ars_kill_o.data[0] = (0 & SGF8_IU.data[0]) & 0x1;
    /* write-back inout operands */
    if (!ars_kill_o.data[0]) *ars_ = *((unsigned *) &ars_o);
    /* update output interface signals */
    MemDataOut8_set();
}

void
SGF8_X(gf8 gr_, unsigned ars_, unsigned art_)
{
    /* operand variables */
    V1_t gr_i;
    V1_t ars_i;
    V1_t art_i;
    /* unused operand variables */
    /* operand kill variables */
    /* one-hot instruction signals */
    /* state variables */
    /* local wire variables */
    /* initialize in/inout operands */
    gr_i.data[0] = gr_;
    ars_i = *((V1_t *) &ars_);
    art_i = *((V1_t *) &art_);
    tie_load_instruction = 0;
    /* semantic statements */
    LSSize.data[0] = 0x1 & 0x1f;
    VAddrBase.data[0] = ars_i.data[0];
    LSIndexed.data[0] = 0x1 & 0x1;
    VAddrIndex.data[0] = art_i.data[0];
    MemDataOut8.data[0] = gr_i.data[0] & 0xff;
    /* write-back inout operands */
    /* update output interface signals */
    MemDataOut8_set();
}

#define SGF8_XU(gr, ars, art) \
        SGF8_XU_func(gr, &(ars), art)

void
SGF8_XU_func(gf8 gr_, unsigned *ars_, unsigned art_)
{
```

```
    /* operand variables */
    V1_t gr_i;
    V1_t ars_o;
    V1_t ars_i;
    V1_t art_i;
    /* unused operand variables */
    /* operand kill variables */
    V1_t ars_kill_o = {{0}};
    /* one-hot instruction signals */
    V1_t SGF8_XU = {{1}};
    /* state variables */
    /* local wire variables */
    /* initialize in/inout operands */
    gr_i.data[0] = gr_;
    ars_i = *((V1_t *) ars_);
    art_i = *((V1_t *) &art_);
    tie_load_instruction = 0;
    /* semantic statements */
    LSSize.data[0] = 0x1 & 0x1f;
    VAddrBase.data[0] = ars_i.data[0];
    LSIndexed.data[0] = 0x1 & 0x1;
    VAddrIndex.data[0] = art_i.data[0];
    MemDataOut8.data[0] = gr_i.data[0] & 0xff;
    VAddrIn_get();
    ars_o.data[0] = VAddrIn.data[0];
    ars_kill_o.data[0] = (0 & SGF8_XU.data[0]) & 0x1;
    /* write-back inout operands */
    if (!ars_kill_o.data[0]) *ars_ = *((unsigned *) &ars_o);
    /* update output interface signals */
    MemDataOut8_set();
}

unsigned
RUR0()
{
    /* operand variables */
    V1_t arr_o;
    /* unused operand variables */
    /* operand kill variables */
    V1_t arr_kill_o = {{0}};
    /* one-hot instruction signals */
    V1_t RUR0 = {{1}};
    /* state variables */
    V1_t gfmod_ps;
    /* local wire variables */
    /* get input state values */
    GetState(gfmod_ps, STATE_gfmod);
    /* initialize in/inout operands */
    tie_load_instruction = 0;
    /* semantic statements */
    arr_o.data[0] = gfmod_ps.data[0];
    arr_kill_o.data[0] = (0 & RUR0.data[0]) & 0x1;
    /* write-back inout operands */
    /* return the output operand */
    return *((unsigned *) &arr_o);
}
```

```
void
WUR0(unsigned art_)
{
    /* operand variables */
    V1_t art_i;
    /* unused operand variables */
    /* operand kill variables */
    /* one-hot instruction signals */
    V1_t WUR0 = {{1}};
    /* state variables */
    V1_t gfmod_ns;
    V1_t gfmod_kill_ns;
    /* local wire variables */
    V1_t tmp0;
    /* initialize in/inout operands */
    art_i = *((V1_t *) &art_);
    tie_load_instruction = 0;
    /* semantic statements */
    tmp0.data[0] = ((art_i.data[0] & 0xff)) & 0xff;
    gfmod_ns.data[0] = (tmp0.data[0]) & 0xff;
    gfmod_kill_ns.data[0] = (0 & WUR0.data[0]) & 0x1;
    /* write-back inout operands */
    /* update out/inout states */
    if (!gfmod_kill_ns.data[0]) SetState(STATE_gfmod, gfmod_ns);
}

#define gf8_loadi(_s, o) ({ \
    gf8 t; \
    gf8 *s = _s; \
    gf8 LGF8_I_return; \
    LGF8_I_return = LGF8_I(*((unsigned *)&(s)), *((int *)&(o))); \
    t = *((gf8 *)&LGF8_I_return); \
    t; \
})

#define gf8_storei(_t, _s, o) ({ \
    gf8 t = _t; \
    gf8 *s = _s; \
    SGF8_I(*((gf8 *)&(t)), *((unsigned *)&(s)), *((int *)&(o))); \
})

#define gf8_move(_r, _s) ({ \
    gf8 r = _r; \
    gf8 s = _s; \
    gf8 GFADD8_return; \
    GFADD8_return = GFADD8(*((gf8 *)&(s)), *((gf8 *)&(0))); \
    r = *((gf8 *)&GFADD8_return); \
})

#ifdef TIE_DEBUG
#undef gf8_loadi
#undef gf8_storei
#undef gf8_move
#undef GFADD8
#undef GFADD8I
#undef GFMULX8
#undef GFRWMOD8
```

```
#undef LGF8_I
#undef SGF8_I
#undef LGF8_IU
#undef SGF8_IU
#undef LGF8_X
#undef SGF8_X
#undef LGF8_XU
#undef SGF8_XU
#undef RUR0
#undef WUR0
#endif
#endif
```

BR.h

```
/*
 * Copyright 1999-2000 Tensilica Inc.
 * These coded instructions, statements, and computer programs are
 * Confidential Proprietary Information of Tensilica Inc. and may not be
 * disclosed to third parties or copied in any form, in whole or in part,
 * without the prior written consent of Tensilica Inc.
 */

#ifndef BR_HEADER
#define BR_HEADER

#ifndef __XTENSA__

typedef unsigned char xtbool;
typedef unsigned char xtbool2;
typedef unsigned char xtbool4;
typedef unsigned char xtbool8;
typedef unsigned short xtbool16;

xtbool
XT_ANDB(xtbool bs, xtbool bt)
{
    return 0x1 & (bs & bt);
}

xtbool
XT_ANDBC(xtbool bs, xtbool bt)
{
    return 0x1 & (bs & !bt);
}

xtbool
XT_ORB(xtbool bs, xtbool bt)
{
    return 0x1 & (bs | bt);
}

xtbool
XT_ORBC(xtbool bs, xtbool bt)
{
    return 0x1 & (bs | !bt);
```

```
}

xtbool
XT_XORB(xtbool bs, xtbool bt)
{
    return 0x1 & (bs ^ bt);
}

xtbool
XT_ANY4(xtbool4 bs4)
{
    return (bs4 & 0xf) != 0;
}

xtbool
XT_ALL4(xtbool4 bs4)
{
    return (bs4 & 0xf) == 0xf;
}

xtbool
XT_ANY8(xtbool8 bs8)
{
    return (bs8 & 0xf) != 0;
}

xtbool
XT_ALL8(xtbool8 bs8)
{
    return (bs8 & 0xf) == 0xf;
}

#endif /* __XTENSA__ */

#endif /* BR_HEADER */
```

gf.v

```
module xmTIE_gf_Regfile(rd0_data_C1, rd0_addr_C0, rd0_width8_C0, rd0_use1_C0,
  rd1_data_C1, rd1_addr_C0, rd1_width8_C0, rd1_use1_C0, rd2_data_C1,
  rd2_addr_C0, rd2_width8_C0, rd2_use1_C0, wd_addr_C0, wd_width8_C0,
  wd_def1_C0, wd_def2_C0, wd_data8_C1, wd_data8_C2, wd_wen_C1, wd_wen_C2,
  Kill_E, KillPipe_W, Stall_R, clk);
    output [7:0] rd0_data_C1;
    input [3:0] rd0_addr_C0;
    input rd0_width8_C0;
    input rd0_use1_C0;
    output [7:0] rd1_data_C1;
    input [3:0] rd1_addr_C0;
    input rd1_width8_C0;
    input rd1_use1_C0;
    output [7:0] rd2_data_C1;
    input [3:0] rd2_addr_C0;
    input rd2_width8_C0;
    input rd2_use1_C0;
    input [3:0] wd_addr_C0;
```

```
input wd_width8_C0;
input wd_def1_C0;
input wd_def2_C0;
input [7:0] wd_data8_C1;
input [7:0] wd_data8_C2;
input wd_wen_C1;
input wd_wen_C2;
input Kill_E;
input KillPipe_W;
output Stall_R;
input clk;


/**********************************************************************
      READ PORT rd0
 **********************************************************************/
// compute the address mask
wire rd0_addr_mask_C0 = 1'd0;

// masked address pipeline
wire rd0_maddr_C0 = 1'd0;

// bank-qualified use
wire rd0_use1_bank0_C0 = (rd0_use1_C0 & (rd0_maddr_C0 == (1'd0 &
rd0_addr_mask_C0)));

// alignment mux for use 1
wire [7:0] rd0_data_bank0_C1;
assign rd0_data_C1[7:0] = rd0_data_bank0_C1;


/**********************************************************************
      READ PORT rd1
 **********************************************************************/
// compute the address mask
wire rd1_addr_mask_C0 = 1'd0;

// masked address pipeline
wire rd1_maddr_C0 = 1'd0;

// bank-qualified use
wire rd1_use1_bank0_C0 = (rd1_use1_C0 & (rd1_maddr_C0 == (1'd0 &
rd1_addr_mask_C0)));

// alignment mux for use 1
wire [7:0] rd1_data_bank0_C1;
assign rd1_data_C1[7:0] = rd1_data_bank0_C1;


/**********************************************************************
      READ PORT rd2
 **********************************************************************/
// compute the address mask
wire rd2_addr_mask_C0 = 1'd0;

// masked address pipeline
wire rd2_maddr_C0 = 1'd0;
```

```
    // bank-qualified use
    wire rd2_use1_bank0_C0 = (rd2_use1_C0 & (rd2_maddr_C0 == (1'd0 &
rd2_addr_mask_C0)));

    // alignment mux for use 1
    wire [7:0] rd2_data_bank0_C1;
    assign rd2_data_C1[7:0] = rd2_data_bank0_C1;


    /*********************************************************************
          WRITE PORT wd
     *********************************************************************/
    // compute the address mask
    wire wd_addr_mask_C0 = 1'd0;

    // bank-qualified write def for port wd
    wire wd_def1_bank0_C0 = (wd_def1_C0 & ((wd_addr_C0 & wd_addr_mask_C0) ==
(1'd0 & wd_addr_mask_C0)));
    wire wd_def2_bank0_C0 = (wd_def2_C0 & ((wd_addr_C0 & wd_addr_mask_C0) ==
(1'd0 & wd_addr_mask_C0)));

    // write mux for def 1
    wire [7:0] wd_wdata_C1;
    assign wd_wdata_C1 = {1{wd_data8_C1[7:0]}};

    // write mux for def 2
    wire [7:0] wd_wdata_C2;
    assign wd_wdata_C2 = {1{wd_data8_C2[7:0]}};

    wire Stall_R0;
    /*********************************************************************
          PIPELINED BANK
     *********************************************************************/
    xmTIE_gf_Regfile_bank TIE_gf_Regfile_bank0(rd0_data_bank0_C1,
        rd0_addr_C0[3:0], rd0_use1_bank0_C0, rd1_data_bank0_C1,
rd1_addr_C0[3:0],
        rd1_use1_bank0_C0, rd2_data_bank0_C1, rd2_addr_C0[3:0],
rd2_use1_bank0_C0,
        wd_addr_C0[3:0], wd_def1_bank0_C0, wd_def2_bank0_C0, wd_wdata_C1[7:0],
        wd_wdata_C2[7:0], wd_wen_C1, wd_wen_C2, Kill_E, KillPipe_W, Stall_R0,
        clk);

    assign Stall_R = Stall_R0 | 1'b0;

endmodule


module xmTIE_gf_Regfile_bank(rd0_data_C1, rd0_addr_C0, rd0_use1_C0,
  rd1_data_C1, rd1_addr_C0, rd1_use1_C0, rd2_data_C1, rd2_addr_C0,
  rd2_use1_C0, wd_addr_C0, wd_def1_C0, wd_def2_C0, wd_data_C1, wd_data_C2,
  wd_wen_C1, wd_wen_C2, Kill_E, KillPipe_W, Stall_R, clk);
    output [7:0] rd0_data_C1;
    input [3:0] rd0_addr_C0;
    input rd0_use1_C0;
    output [7:0] rd1_data_C1;
```

```
input [3:0] rd1_addr_C0;
input rd1_use1_C0;
output [7:0] rd2_data_C1;
input [3:0] rd2_addr_C0;
input rd2_use1_C0;
input [3:0] wd_addr_C0;
input wd_def1_C0;
input wd_def2_C0;
input [7:0] wd_data_C1;
input [7:0] wd_data_C2;
input wd_wen_C1;
input wd_wen_C2;
input Kill_E;
input KillPipe_W;
output Stall_R;
input clk;

wire rd0_use2_C0 = 1'd0;
wire rd1_use2_C0 = 1'd0;
wire rd2_use2_C0 = 1'd0;

wire kill_C0 = KillPipe_W;
wire kill_C1 = KillPipe_W | Kill_E;
wire kill_C2 = KillPipe_W;
wire kill_C3 = KillPipe_W;

// write definition pipeline
wire wd_ns_def1_C0 = wd_def1_C0 & 1'b1 & ~kill_C0;
wire wd_def1_C1;
xtdelay1 #(1) iwd_def1_C1(wd_def1_C1, wd_ns_def1_C0, clk);
wire wd_ns_def2_C0 = wd_def2_C0 & 1'b1 & ~kill_C0;
wire wd_def2_C1;
xtdelay1 #(1) iwd_def2_C1(wd_def2_C1, wd_ns_def2_C0, clk);
wire wd_ns_def2_C1 = wd_def2_C1 & wd_wen_C1 & ~kill_C1;
wire wd_def2_C2;
xtdelay1 #(1) iwd_def2_C2(wd_def2_C2, wd_ns_def2_C1, clk);

// write enable pipeline
wire wd_we_C2;
wire wd_we_C3;
wire wd_ns_we_C1 = (1'd0 | (wd_def1_C1 & wd_wen_C1)) & ~kill_C1;
wire wd_ns_we_C2 = (wd_we_C2 | (wd_def2_C2 & wd_wen_C2)) & ~kill_C2;
wire wd_ns_we_C3 = (wd_we_C3 | (1'd0 & 1'd0)) & ~kill_C3;
xtdelay1 #(1) iwd_we_C2(wd_we_C2, wd_ns_we_C1, clk);
xtdelay1 #(1) iwd_we_C3(wd_we_C3, wd_ns_we_C2, clk);

// write address pipeline
wire [3:0] wd_addr_C1;
wire [3:0] wd_addr_C2;
wire [3:0] wd_addr_C3;
xtdelay1 #(4) iwd_addr_C1(wd_addr_C1, wd_addr_C0, clk);
xtdelay1 #(4) iwd_addr_C2(wd_addr_C2, wd_addr_C1, clk);
xtdelay1 #(4) iwd_addr_C3(wd_addr_C3, wd_addr_C2, clk);

// write data pipeline
wire [7:0] wd_result_C2;
wire [7:0] wd_result_C3;
```

```
    wire [7:0] wd_mux_C1 = wd_data_C1;
    wire [7:0] wd_mux_C2 = wd_def2_C2 ? wd_data_C2 : wd_result_C2;
    xtdelay1 #(8) iwd_result_C2(wd_result_C2, wd_mux_C1, clk);
    xtdelay1 #(8) iwd_result_C3(wd_result_C3, wd_mux_C2, clk);

    wire [7:0] rd0_data_C0;
    wire [7:0] rd1_data_C0;
    wire [7:0] rd2_data_C0;

    // Read bypass controls for port rd0
    wire bypass_data_rd0_C0_wd_C1 = (wd_addr_C1 == rd0_addr_C0) & wd_def1_C1 &
wd_wen_C1 & ~kill_C1;
    wire bypass_data_rd0_C0_wd_C2 = (wd_addr_C2 == rd0_addr_C0) & wd_def2_C2 &
wd_wen_C2 & ~kill_C2;
    wire bypass_result_rd0_C0_wd_C2 = (wd_addr_C2 == rd0_addr_C0) & wd_we_C2 &
~kill_C2;
    wire bypass_result_rd0_C0_wd_C3 = (wd_addr_C3 == rd0_addr_C0) & wd_we_C3 &
~kill_C3;

    // Read bypass for port rd0 use 1
    wire [7:0] rd0_mux_result_C0;
    xtmux3p #(8) m0(rd0_mux_result_C0, wd_result_C2, wd_result_C3, rd0_data_C0,
bypass_result_rd0_C0_wd_C2, bypass_result_rd0_C0_wd_C3);
    wire [7:0] rd0_mux_C0;
    wire [1:0] rd0_mux_C0_sel =
        bypass_data_rd0_C0_wd_C1 ? 2'd1 :
        bypass_data_rd0_C0_wd_C2 ? 2'd2 :
        bypass_result_rd0_C0_wd_C2 ? 2'd0 :
        bypass_result_rd0_C0_wd_C3 ? 2'd0 :
        2'd0;
    xtmux3e #(8) m1(rd0_mux_C0, rd0_mux_result_C0, wd_data_C1, wd_data_C2,
rd0_mux_C0_sel);
    xtdelay1 #(8) ird0_data_C1(rd0_data_C1, rd0_mux_C0, clk);

    // Read bypass controls for port rd1
    wire bypass_data_rd1_C0_wd_C1 = (wd_addr_C1 == rd1_addr_C0) & wd_def1_C1 &
wd_wen_C1 & ~kill_C1;
    wire bypass_data_rd1_C0_wd_C2 = (wd_addr_C2 == rd1_addr_C0) & wd_def2_C2 &
wd_wen_C2 & ~kill_C2;
    wire bypass_result_rd1_C0_wd_C2 = (wd_addr_C2 == rd1_addr_C0) & wd_we_C2 &
~kill_C2;
    wire bypass_result_rd1_C0_wd_C3 = (wd_addr_C3 == rd1_addr_C0) & wd_we_C3 &
~kill_C3;

    // Read bypass for port rd1 use 1
    wire [7:0] rd1_mux_result_C0;
    xtmux3p #(8) m2(rd1_mux_result_C0, wd_result_C2, wd_result_C3, rd1_data_C0,
bypass_result_rd1_C0_wd_C2, bypass_result_rd1_C0_wd_C3);
    wire [7:0] rd1_mux_C0;
    wire [1:0] rd1_mux_C0_sel =
        bypass_data_rd1_C0_wd_C1 ? 2'd1 :
        bypass_data_rd1_C0_wd_C2 ? 2'd2 :
        bypass_result_rd1_C0_wd_C2 ? 2'd0 :
        bypass_result_rd1_C0_wd_C3 ? 2'd0 :
        2'd0;
    xtmux3e #(8) m3(rd1_mux_C0, rd1_mux_result_C0, wd_data_C1, wd_data_C2,
rd1_mux_C0_sel);
```

```
    xtdelay1 #(8) ird1_data_C1(rd1_data_C1, rd1_mux_C0, clk);

    // Read bypass controls for port rd2
    wire bypass_data_rd2_C0_wd_C1 = (wd_addr_C1 == rd2_addr_C0) & wd_def1_C1 &
wd_wen_C1 & ~kill_C1;
    wire bypass_data_rd2_C0_wd_C2 = (wd_addr_C2 == rd2_addr_C0) & wd_def2_C2 &
wd_wen_C2 & ~kill_C2;
    wire bypass_result_rd2_C0_wd_C2 = (wd_addr_C2 == rd2_addr_C0) & wd_we_C2 &
~kill_C2;
    wire bypass_result_rd2_C0_wd_C3 = (wd_addr_C3 == rd2_addr_C0) & wd_we_C3 &
~kill_C3;

    // Read bypass for port rd2 use 1
    wire [7:0] rd2_mux_result_C0;
    xtmux3p #(8) m4(rd2_mux_result_C0, wd_result_C2, wd_result_C3, rd2_data_C0,
bypass_result_rd2_C0_wd_C2, bypass_result_rd2_C0_wd_C3);
    wire [7:0] rd2_mux_C0;
    wire [1:0] rd2_mux_C0_sel =
        bypass_data_rd2_C0_wd_C1 ? 2'd1 :
        bypass_data_rd2_C0_wd_C2 ? 2'd2 :
        bypass_result_rd2_C0_wd_C2 ? 2'd0 :
        bypass_result_rd2_C0_wd_C3 ? 2'd0 :
        2'd0;
    xtmux3e #(8) m5(rd2_mux_C0, rd2_mux_result_C0, wd_data_C1, wd_data_C2,
rd2_mux_C0_sel);
    xtdelay1 #(8) ird2_data_C1(rd2_data_C1, rd2_mux_C0, clk);

    assign Stall_R =
        ((wd_addr_C1 == rd0_addr_C0) & (
            (rd0_use1_C0 & (wd_ns_def2_C1)))) |
        ((wd_addr_C1 == rd1_addr_C0) & (
            (rd1_use1_C0 & (wd_ns_def2_C1)))) |
        ((wd_addr_C1 == rd2_addr_C0) & (
            (rd2_use1_C0 & (wd_ns_def2_C1)))) |
        1'b0;

    // register file core
    xtregfile_3R1W_16 #(8) icore(rd0_data_C0, rd0_addr_C0, rd1_data_C0,
        rd1_addr_C0, rd2_data_C0, rd2_addr_C0, wd_result_C3, wd_addr_C3,
        wd_ns_we_C3, clk);
endmodule


module xmTIE_gfmod_State(ps_data_C1, ps_width8_C0, ps_use1_C0, ns_width8_C0,
  ns_def1_C0, ns_data8_C1, ns_wen_C1, Kill_E, KillPipe_W, Stall_R, clk);
    output [7:0] ps_data_C1;
    input ps_width8_C0;
    input ps_use1_C0;
    input ns_width8_C0;
    input ns_def1_C0;
    input [7:0] ns_data8_C1;
    input ns_wen_C1;
    input Kill_E;
    input KillPipe_W;
    output Stall_R;
    input clk;
```

```
    wire ps_addr_C0 = 1'd0;
    wire ns_addr_C0 = 1'd0;
    wire ns_wen_C2 = 1'd1;

    /*********************************************************************
          READ PORT ps
     *********************************************************************/
    // compute the address mask
    wire ps_addr_mask_C0 = 1'd0;

    // masked address pipeline
    wire ps_maddr_C0 = 1'd0;

    // bank-qualified use
    wire ps_use1_bank0_C0 = (ps_use1_C0 & (ps_maddr_C0 == (1'd0 &
ps_addr_mask_C0)));

    // alignment mux for use 1
    wire [7:0] ps_data_bank0_C1;
    assign ps_data_C1[7:0] = ps_data_bank0_C1;


    /*********************************************************************
          WRITE PORT ns
     *********************************************************************/
    // compute the address mask
    wire ns_addr_mask_C0 = 1'd0;

    // bank-qualified write def for port ns
    wire ns_def1_bank0_C0 = (ns_def1_C0 & ((ns_addr_C0 & ns_addr_mask_C0) ==
(1'd0 & ns_addr_mask_C0)));

    // write mux for def 1
    wire [7:0] ns_wdata_C1;
    assign ns_wdata_C1 = {1{ns_data8_C1[7:0]}};


    wire Stall_R0;
    /*********************************************************************
          PIPELINED BANK
     *********************************************************************/
    xmTIE_gfmod_State_bank TIE_gfmod_State_bank0(ps_data_bank0_C1,
        ps_use1_bank0_C0, ns_def1_bank0_C0, ns_wdata_C1[7:0], ns_wen_C1,
        ns_wen_C2, Kill_E, KillPipe_W, Stall_R0, clk);

    assign Stall_R = Stall_R0 | 1'b0;

endmodule


module xmTIE_gfmod_State_bank(ps_data_C1, ps_use1_C0, ns_def1_C0, ns_data_C1,
  ns_wen_C1, ns_wen_C2, Kill_E, KillPipe_W, Stall_R, clk);
    output [7:0] ps_data_C1;
    input ps_use1_C0;
    input ns_def1_C0;
    input [7:0] ns_data_C1;
    input ns_wen_C1;
```

```
input ns_wen_C2;
input Kill_E;
input KillPipe_W;
output Stall_R;
input clk;

wire ps_addr_C0 = 1'd0;
wire ps_use2_C0 = 1'd0;
wire ns_addr_C0 = 1'd0;
wire ns_def2_C0 = 1'd0;
wire [7:0] ns_data_C2 = 0;

wire kill_C0 = KillPipe_W;
wire kill_C1 = KillPipe_W | Kill_E;
wire kill_C2 = KillPipe_W;
wire kill_C3 = KillPipe_W;

// write definition pipeline
wire ns_ns_def1_C0 = ns_def1_C0 & 1'b1 & ~kill_C0;
wire ns_def1_C1;
xtdelay1 #(1) ins_def1_C1(ns_def1_C1, ns_ns_def1_C0, clk);
wire ns_ns_def2_C0 = 1'd0;
wire ns_def2_C1 = 1'd0;
wire ns_ns_def2_C1 = 1'd0;
wire ns_def2_C2 = 1'd0;

// write enable pipeline
wire ns_we_C2;
wire ns_we_C3;
wire ns_ns_we_C1 = (1'd0 | (ns_def1_C1 & ns_wen_C1)) & ~kill_C1;
wire ns_ns_we_C2 = (ns_we_C2 | (ns_def2_C2 & ns_wen_C2)) & ~kill_C2;
wire ns_ns_we_C3 = (ns_we_C3 | (1'd0 & 1'd0)) & ~kill_C3;
xtdelay1 #(1) ins_we_C2(ns_we_C2, ns_ns_we_C1, clk);
xtdelay1 #(1) ins_we_C3(ns_we_C3, ns_ns_we_C2, clk);

// write address pipeline
wire  ns_addr_C1;
wire  ns_addr_C2;
wire  ns_addr_C3;
assign ns_addr_C1 = 1'd0;
assign ns_addr_C2 = 1'd0;
assign ns_addr_C3 = 1'd0;

// write data pipeline
wire [7:0] ns_result_C2;
wire [7:0] ns_result_C3;
wire [7:0] ns_mux_C1 = ns_data_C1;
wire [7:0] ns_mux_C2 = ns_def2_C2 ? ns_data_C2 : ns_result_C2;
xtdelay1 #(8) ins_result_C2(ns_result_C2, ns_mux_C1, clk);
xtdelay1 #(8) ins_result_C3(ns_result_C3, ns_mux_C2, clk);

wire [7:0] ps_data_C0;

// Read bypass controls for port ps
wire bypass_data_ps_C0_ns_C1 = (ns_addr_C1 == ps_addr_C0) & ns_def1_C1 &
ns_wen_C1 & ~kill_C1;
```

```
    wire bypass_result_ps_C0_ns_C2 = (ns_addr_C2 == ps_addr_C0) & ns_we_C2 &
~kill_C2;
    wire bypass_result_ps_C0_ns_C3 = (ns_addr_C3 == ps_addr_C0) & ns_we_C3 &
~kill_C3;

    // Read bypass for port ps use 1
    wire [7:0] ps_mux_result_C0;
    xtmux3p #(8) m6(ps_mux_result_C0, ns_result_C2, ns_result_C3, ps_data_C0,
bypass_result_ps_C0_ns_C2, bypass_result_ps_C0_ns_C3);
    wire [7:0] ps_mux_C0;
    wire [0:0] ps_mux_C0_sel =
        bypass_data_ps_C0_ns_C1 ? 1'd1 :
        bypass_result_ps_C0_ns_C2 ? 1'd0 :
        bypass_result_ps_C0_ns_C3 ? 1'd0 :
        1'd0;
    xtmux2e #(8) m7(ps_mux_C0, ps_mux_result_C0, ns_data_C1, ps_mux_C0_sel);
    xtdelay1 #(8) ips_data_C1(ps_data_C1, ps_mux_C0, clk);

    assign Stall_R =
        ((ns_addr_C1 == ps_addr_C0) & (
            (ps_use1_C0 & (ns_ns_def2_C1)))) |
        1'b0;

    // register file core
    xtregfile_1R1W_1 #(8) icore(ps_data_C0, ns_result_C3, ns_ns_we_C3, clk);
endmodule


module xmTIE_decoder (
GFADD8,
GFADD8I,
GFMULX8,
GFRWMOD8,
LGF8_I,
SGF8_I,
LGF8_IU,
SGF8_IU,
LGF8_X,
SGF8_X,
LGF8_XU,
SGF8_XU,
RUR0,
WUR0,
imm4,
imm8,
art_use,
art_def,
ars_use,
ars_def,
arr_use,
arr_def,
br_use,
br_def,
bs_use,
bs_def,
bt_use,
bt_def,
```

```
bs4_use,
bs4_def,
bs8_use,
bs8_def,
gr_use,
gr_def,
gs_use,
gs_def,
gt_use,
gt_def,
gfmod_use1,
gfmod_def1,
AR_rd0_use1,
AR_rd0_width32,
AR_rd1_use1,
AR_rd1_width32,
AR_wd_def1,
AR_wd_width32,
gf_rd0_addr,
gf_rd0_use1,
gf_rd0_width8,
gf_rd1_addr,
gf_rd1_use1,
gf_rd1_width8,
gf_rd2_addr,
gf_rd2_use1,
gf_rd2_width8,
gf_wd_addr,
gf_wd_def2,
gf_wd_def1,
gf_wd_width8,
gf1_semantic,
gf4_semantic,
gf2_semantic,
gf3_semantic,
lgf_semantic,
sgf_semantic,
RUR0_semantic,
WUR0_semantic,
load_instruction,
store_instruction,
TIE_Inst,
Inst
);
output GFADD8;
output GFADD8I;
output GFMULX8;
output GFRWMOD8;
output LGF8_I;
output SGF8_I;
output LGF8_IU;
output SGF8_IU;
output LGF8_X;
output SGF8_X;
output LGF8_XU;
output SGF8_XU;
output RUR0;
```

```
output WURO;
output [31:0] imm4;
output [7:0] imm8;
output art_use;
output art_def;
output ars_use;
output ars_def;
output arr_use;
output arr_def;
output br_use;
output br_def;
output bs_use;
output bs_def;
output bt_use;
output bt_def;
output bs4_use;
output bs4_def;
output bs8_use;
output bs8_def;
output gr_use;
output gr_def;
output gs_use;
output gs_def;
output gt_use;
output gt_def;
output gfmod_use1;
output gfmod_def1;
output AR_rd0_use1;
output AR_rd0_width32;
output AR_rd1_use1;
output AR_rd1_width32;
output AR_wd_def1;
output AR_wd_width32;
output [3:0] gf_rd0_addr;
output gf_rd0_use1;
output gf_rd0_width8;
output [3:0] gf_rd1_addr;
output gf_rd1_use1;
output gf_rd1_width8;
output [3:0] gf_rd2_addr;
output gf_rd2_use1;
output gf_rd2_width8;
output [3:0] gf_wd_addr;
output gf_wd_def2;
output gf_wd_def1;
output gf_wd_width8;
output gf1_semantic;
output gf4_semantic;
output gf2_semantic;
output gf3_semantic;
output lgf_semantic;
output sgf_semantic;
output RURO_semantic;
output WURO_semantic;
output load_instruction;
output store_instruction;
output TIE_Inst;
```

```
input [23:0] Inst;

wire [3:0] op2 = {Inst[23:20]};
wire [3:0] op1 = {Inst[19:16]};
wire [3:0] op0 = {Inst[3:0]};
wire QRST = (op0==4'b0000);
wire CUST0 = (op1==4'b0110) & QRST;
assign GFADD8 = (op2==4'b0000) & CUST0;
assign GFADD8I = (op2==4'b0100) & CUST0;
assign GFMULX8 = (op2==4'b0001) & CUST0;
assign GFRWMOD8 = (op2==4'b0010) & CUST0;
wire [3:0] r = {Inst[15:12]};
wire LSCI = (op0==4'b0011);
assign LGF8_I = (r==4'b0000) & LSCI;
assign SGF8_I = (r==4'b0001) & LSCI;
assign LGF8_IU = (r==4'b0010) & LSCI;
assign SGF8_IU = (r==4'b0011) & LSCI;
wire LSCX = (op1==4'b1000) & QRST;
assign LGF8_X = (op2==4'b0000) & LSCX;
assign SGF8_X = (op2==4'b0001) & LSCX;
assign LGF8_XU = (op2==4'b0010) & LSCX;
assign SGF8_XU = (op2==4'b0011) & LSCX;
wire [3:0] s = {Inst[11:8]};
wire [3:0] t = {Inst[7:4]};
wire [7:0] st = {s,t};
wire RST3 = (op1==4'b0011) & QRST;
wire RUR = (op2==4'b1110) & RST3;
assign RUR0 = (st==8'b00000000) & RUR;
wire [7:0] sr = {r,s};
wire WUR = (op2==4'b1111) & RST3;
assign WUR0 = (sr==8'b00000000) & WUR;
assign gfmod_use1 = GFMULX8 | GFRWMOD8 | RUR0 | 1'b0;
assign gfmod_def1 = GFRWMOD8 | WUR0 | 1'b0;
assign AR_rd0_use1 = 1'b0
    | LGF8_I
    | SGF8_I
    | LGF8_IU
    | SGF8_IU
    | LGF8_X
    | SGF8_X
    | LGF8_XU
    | SGF8_XU;
assign AR_rd0_width32 = 1'b0;
assign AR_rd1_use1 = 1'b0
    | LGF8_X
    | SGF8_X
    | LGF8_XU
    | SGF8_XU
    | WUR0;
assign AR_rd1_width32 = 1'b0;
assign AR_wd_def1 = 1'b0
    | LGF8_IU
    | SGF8_IU
    | LGF8_XU
    | SGF8_XU
    | RUR0;
assign AR_wd_width32 = 1'b0;
```

```
assign gf_rd0_usel = 1'b0
     | GFADD8
     | GFADD8I
     | GFMULX8;
assign gf_rd0_width8 = 1'b0;
assign gf_rd1_usel = 1'b0
     | GFADD8
     | GFRWMOD8
     | SGF8_I
     | SGF8_IU;
assign gf_rd1_width8 = 1'b0;
assign gf_rd2_usel = 1'b0
     | SGF8_X
     | SGF8_XU;
assign gf_rd2_width8 = 1'b0;
assign gf_wd_def2 = 1'b0
     | LGF8_I
     | LGF8_IU
     | LGF8_X
     | LGF8_XU;
assign gf_wd_def1 = 1'b0
     | GFADD8
     | GFADD8I
     | GFMULX8
     | GFRWMOD8;
assign gf_wd_width8 = 1'b0;
assign art_def = 1'b0;
assign art_use = LGF8_X | SGF8_X | LGF8_XU | SGF8_XU | WUR0 | 1'b0;
assign ars_def = LGF8_IU | SGF8_IU | LGF8_XU | SGF8_XU | 1'b0;
assign ars_use = LGF8_I | SGF8_I | LGF8_IU | SGF8_IU | LGF8_X | SGF8_X |
LGF8_XU | SGF8_XU | 1'b0;
assign arr_def = RUR0 | 1'b0;
assign arr_use = 1'b0;
assign br_def = 1'b0;
assign br_use = 1'b0;
assign bs_def = 1'b0;
assign bs_use = 1'b0;
assign bt_def = 1'b0;
assign bt_use = 1'b0;
assign bs4_def = 1'b0;
assign bs4_use = 1'b0;
assign bs8_def = 1'b0;
assign bs8_use = 1'b0;
assign gr_def = GFADD8 | GFADD8I | GFMULX8 | LGF8_X | LGF8_XU | 1'b0;
assign gr_use = SGF8_X | SGF8_XU | 1'b0;
assign gs_def = 1'b0;
assign gs_use = GFADD8 | GFADD8I | GFMULX8 | 1'b0;
assign gt_def = GFRWMOD8 | LGF8_I | LGF8_IU | 1'b0;
assign gt_use = GFADD8 | GFRWMOD8 | SGF8_I | SGF8_IU | 1'b0;
wire [3:0] gr_addr = r;
wire [3:0] gs_addr = s;
wire [3:0] gt_addr = t;
assign gf_wd_addr = 4'b0
    | {4{gr_def}} & gr_addr
    | {4{gt_def}} & gt_addr;
assign gf_rd0_addr = gs_addr;
assign gf_rd1_addr = gt_addr;
```

```
assign gf_rd2_addr = gr_addr;
assign gf1_semantic = GFADD8 | 1'b0;
assign gf4_semantic = GFADD8I | 1'b0;
assign gf2_semantic = GFMULX8 | 1'b0;
assign gf3_semantic = GFRWMOD8 | 1'b0;
assign lgf_semantic = LGF8_I | LGF8_IU | LGF8_X | LGF8_XU | 1'b0;
assign sgf_semantic = SGF8_I | SGF8_IU | SGF8_X | SGF8_XU | 1'b0;
assign RUR0_semantic = RUR0 | 1'b0;
assign WUR0_semantic = WUR0 | 1'b0;
assign imm4 = t;
wire [7:0] imm8 = {Inst[23:16]};
assign load_instruction = 1'b0
    | LGF8_I
    | LGF8_IU
    | LGF8_X
    | LGF8_XU;
assign store_instruction = 1'b0
    | SGF8_I
    | SGF8_IU
    | SGF8_X
    | SGF8_XU;
assign TIE_Inst = 1'b0
    | GFADD8
    | GFADD8I
    | GFMULX8
    | GFRWMOD8
    | LGF8_I
    | SGF8_I
    | LGF8_IU
    | SGF8_IU
    | LGF8_X
    | SGF8_X
    | LGF8_XU
    | SGF8_XU
    | RUR0
    | WUR0;
endmodule

module xmTIE_gf1 (
GFADD8_C0,
gr_o_C1,
gr_kill_C1,
gs_i_C1,
gt_i_C1,
clk
);
input GFADD8_C0;
output [7:0] gr_o_C1;
output gr_kill_C1;
input [7:0] gs_i_C1;
input [7:0] gt_i_C1;
input clk;
assign gr_o_C1 = (gs_i_C1) ^ (gt_i_C1);
wire GFADD8_C1;
xtdelay1 #(1) iGFADD8_C1(.xtin(GFADD8_C0), .xtout(GFADD8_C1), .clk(clk));
assign gr_kill_C1 = (1'b0) & (GFADD8_C1);
endmodule
```

```
module xmTIE_gf4 (
GFADD8I_C0,
gr_o_C1,
gr_kill_C1,
gs_i_C1,
imm4_C0,
clk
);
input GFADD8I_C0;
output [7:0] gr_o_C1;
output gr_kill_C1;
input [7:0] gs_i_C1;
input [31:0] imm4_C0;
input clk;
wire [31:0] imm4_C1;
xtdelay1 #(32) iimm4_C1(.xtin(imm4_C0), .xtout(imm4_C1), .clk(clk));
assign gr_o_C1 = (gs_i_C1) ^ (imm4_C1);
wire GFADD8I_C1;
xtdelay1 #(1) iGFADD8I_C1(.xtin(GFADD8I_C0), .xtout(GFADD8I_C1), .clk(clk));
assign gr_kill_C1 = (1'b0) & (GFADD8I_C1);
endmodule

module xmTIE_gf2 (
GFMULX8_C0,
gr_o_C1,
gr_kill_C1,
gs_i_C1,
gfmod_ps_C1,
clk
);
input GFMULX8_C0;
output [7:0] gr_o_C1;
output gr_kill_C1;
input [7:0] gs_i_C1;
input [7:0] gfmod_ps_C1;
input clk;
assign gr_o_C1 = (gs_i_C1[7]) ? (({gs_i_C1[6:0], 1'b0}) ^ (gfmod_ps_C1)) :
({gs_i_C1[6:0], 1'b0});
wire GFMULX8_C1;
xtdelay1 #(1) iGFMULX8_C1(.xtin(GFMULX8_C0), .xtout(GFMULX8_C1), .clk(clk));
assign gr_kill_C1 = (1'b0) & (GFMULX8_C1);
endmodule

module xmTIE_gf3 (
GFRWMOD8_C0,
gt_i_C1,
gt_o_C1,
gt_kill_C1,
gfmod_ps_C1,
gfmod_ns_C1,
gfmod_kill_C1,
clk
);
input GFRWMOD8_C0;
input [7:0] gt_i_C1;
output [7:0] gt_o_C1;
```

```
output gt_kill_C1;
input [7:0] gfmod_ps_C1;
output [7:0] gfmod_ns_C1;
output gfmod_kill_C1;
input clk;
wire [7:0] t1_C1;
assign t1_C1 = gt_i_C1;
wire [7:0] t2_C1;
assign t2_C1 = gfmod_ps_C1;
assign gfmod_ns_C1 = t1_C1;
assign gt_o_C1 = t2_C1;
wire GFRWMOD8_C1;
xtdelay1 #(1) iGFRWMOD8_C1(.xtin(GFRWMOD8_C0), .xtout(GFRWMOD8_C1), .clk(clk));
assign gfmod_kill_C1 = (1'b0) & (GFRWMOD8_C1);
assign gt_kill_C1 = (1'b0) & (GFRWMOD8_C1);
endmodule

module xmTIE_lgf (
LGF8_I_C0,
LGF8_IU_C0,
LGF8_X_C0,
LGF8_XU_C0,
gt_o_C2,
gt_kill_C2,
ars_i_C1,
ars_o_C1,
ars_kill_C1,
imm8_C0,
gr_o_C2,
gr_kill_C2,
art_i_C1,
MemDataIn8_C2,
VAddrIn_C1,
LSSize_C0,
VAddrBase_C1,
VAddrIndex_C1,
VAddrOffset_C0,
LSIndexed_C0,
clk
);
input LGF8_I_C0;
input LGF8_IU_C0;
input LGF8_X_C0;
input LGF8_XU_C0;
output [7:0] gt_o_C2;
output gt_kill_C2;
input [31:0] ars_i_C1;
output [31:0] ars_o_C1;
output ars_kill_C1;
input [7:0] imm8_C0;
output [7:0] gr_o_C2;
output gr_kill_C2;
input [31:0] art_i_C1;
input [7:0] MemDataIn8_C2;
input [31:0] VAddrIn_C1;
output [4:0] LSSize_C0;
output [31:0] VAddrBase_C1;
```

```
output [31:0] VAddrIndex_C1;
output [31:0] VAddrOffset_C0;
output LSIndexed_C0;
input clk;
wire indexed_C0;
assign indexed_C0 = (LGF8_X_C0) | (LGF8_XU_C0);
assign LSSize_C0 = 32'h1;
assign VAddrBase_C1 = ars_i_C1;
assign LSIndexed_C0 = indexed_C0;
assign VAddrOffset_C0 = imm8_C0;
assign VAddrIndex_C1 = art_i_C1;
assign gt_o_C2 = MemDataIn8_C2;
assign gr_o_C2 = MemDataIn8_C2;
assign ars_o_C1 = VAddrIn_C1;
wire LGF8_I_C2;
xtdelay2 #(1) iLGF8_I_C2(.xtin(LGF8_I_C0), .xtout(LGF8_I_C2), .clk(clk));
wire LGF8_IU_C2;
xtdelay2 #(1) iLGF8_IU_C2(.xtin(LGF8_IU_C0), .xtout(LGF8_IU_C2), .clk(clk));
assign gt_kill_C2 = (1'b0) & ((LGF8_I_C2) | (LGF8_IU_C2));
wire LGF8_IU_C1;
xtdelay1 #(1) iLGF8_IU_C1(.xtin(LGF8_IU_C0), .xtout(LGF8_IU_C1), .clk(clk));
wire LGF8_XU_C1;
xtdelay1 #(1) iLGF8_XU_C1(.xtin(LGF8_XU_C0), .xtout(LGF8_XU_C1), .clk(clk));
assign ars_kill_C1 = (1'b0) & ((LGF8_IU_C1) | (LGF8_XU_C1));
wire LGF8_X_C2;
xtdelay2 #(1) iLGF8_X_C2(.xtin(LGF8_X_C0), .xtout(LGF8_X_C2), .clk(clk));
wire LGF8_XU_C2;
xtdelay2 #(1) iLGF8_XU_C2(.xtin(LGF8_XU_C0), .xtout(LGF8_XU_C2), .clk(clk));
assign gr_kill_C2 = (1'b0) & ((LGF8_X_C2) | (LGF8_XU_C2));
endmodule

module xmTIE_sgf (
SGF8_I_C0,
SGF8_IU_C0,
SGF8_X_C0,
SGF8_XU_C0,
gt_i_C1,
ars_i_C1,
ars_o_C1,
ars_kill_C1,
imm8_C0,
gr_i_C1,
art_i_C1,
VAddrIn_C1,
LSSize_C0,
MemDataOut8_C1,
VAddrBase_C1,
VAddrIndex_C1,
VAddrOffset_C0,
LSIndexed_C0,
clk
);
input SGF8_I_C0;
input SGF8_IU_C0;
input SGF8_X_C0;
input SGF8_XU_C0;
input [7:0] gt_i_C1;
```

```
input [31:0] ars_i_C1;
output [31:0] ars_o_C1;
output ars_kill_C1;
input [7:0] imm8_C0;
input [7:0] gr_i_C1;
input [31:0] art_i_C1;
input [31:0] VAddrIn_C1;
output [4:0] LSSize_C0;
output [7:0] MemDataOut8_C1;
output [31:0] VAddrBase_C1;
output [31:0] VAddrIndex_C1;
output [31:0] VAddrOffset_C0;
output LSIndexed_C0;
input clk;
wire indexed_C0;
assign indexed_C0 = (SGF8_X_C0) | (SGF8_XU_C0);
assign LSSize_C0 = 32'h1;
assign VAddrBase_C1 = ars_i_C1;
assign LSIndexed_C0 = indexed_C0;
assign VAddrOffset_C0 = imm8_C0;
assign VAddrIndex_C1 = art_i_C1;
wire SGF8_X_C1;
xtdelay1 #(1) iSGF8_X_C1(.xtin(SGF8_X_C0), .xtout(SGF8_X_C1), .clk(clk));
wire SGF8_XU_C1;
xtdelay1 #(1) iSGF8_XU_C1(.xtin(SGF8_XU_C0), .xtout(SGF8_XU_C1), .clk(clk));
assign MemDataOut8_C1 = ((SGF8_X_C1) | (SGF8_XU_C1)) ? (gr_i_C1) : (gt_i_C1);
assign ars_o_C1 = VAddrIn_C1;
wire SGF8_IU_C1;
xtdelay1 #(1) iSGF8_IU_C1(.xtin(SGF8_IU_C0), .xtout(SGF8_IU_C1), .clk(clk));
assign ars_kill_C1 = (1'b0) & ((SGF8_IU_C1) | (SGF8_XU_C1));
endmodule

module xmTIE_RUR0 (
RUR0_C0,
arr_o_C1,
arr_kill_C1,
gfmod_ps_C1,
clk
);
input RUR0_C0;
output [31:0] arr_o_C1;
output arr_kill_C1;
input [7:0] gfmod_ps_C1;
input clk;
assign arr_o_C1 = {gfmod_ps_C1};
wire RUR0_C1;
xtdelay1 #(1) iRUR0_C1(.xtin(RUR0_C0), .xtout(RUR0_C1), .clk(clk));
assign arr_kill_C1 = (1'b0) & (RUR0_C1);
endmodule

module xmTIE_WUR0 (
WUR0_C0,
art_i_C1,
gfmod_ns_C1,
gfmod_kill_C1,
clk
);
```

```
input WUR0_C0;
input [31:0] art_i_C1;
output [7:0] gfmod_ns_C1;
output gfmod_kill_C1;
input clk;
assign gfmod_ns_C1 = {art_i_C1[7:0]};
wire WUR0_C1;
xtdelay1 #(1) iWUR0_C1(.xtin(WUR0_C0), .xtout(WUR0_C1), .clk(clk));
assign gfmod_kill_C1 = (1'b0) & (WUR0_C1);
endmodule

module xmTIE (
TIE_inst_R,
TIE_asRead_R,
TIE_atRead_R,
TIE_atWrite_R,
TIE_arWrite_R,
TIE_asWrite_R,
TIE_aWriteM_R,
TIE_aDataKill_E,
TIE_aWriteData_E,
TIE_aDataKill_M,
TIE_aWriteData_M,
TIE_Load_R,
TIE_Store_R,
TIE_LSSize_R,
TIE_LSIndexed_R,
TIE_LSOffset_R,
TIE_MemLoadData_M,
TIE_MemStoreData8_E,
TIE_MemStoreData16_E,
TIE_MemStoreData32_E,
TIE_MemStoreData64_E,
TIE_MemStoreData128_E,
TIE_Stall_R,
TIE_Exception_E,
TIE_ExcCause_E,
TIE_bsRead_R,
TIE_btRead_R,
TIE_btWrite_R,
TIE_brWrite_R,
TIE_bsWrite_R,
TIE_bsReadSize_R,
TIE_btReadSize_R,
TIE_bWriteSize_R,
TIE_bsReadData_E,
TIE_btReadData_E,
TIE_bWriteData1_E,
TIE_bWriteData2_E,
TIE_bWriteData4_E,
TIE_bWriteData8_E,
TIE_bWriteData16_E,
TIE_bDataKill_E,
CPEnable,
Instr_R,
SBus_E,
TBus_E,
```

```
MemOpAddr_E,
Kill_E,
Except_W,
Replay_W,
G1WCLK,
Reset
);
output TIE_inst_R;
output TIE_asRead_R;
output TIE_atRead_R;
output TIE_atWrite_R;
output TIE_arWrite_R;
output TIE_asWrite_R;
output TIE_aWriteM_R;
output TIE_aDataKill_E;
output [31:0] TIE_aWriteData_E;
output TIE_aDataKill_M;
output [31:0] TIE_aWriteData_M;
output TIE_Load_R;
output TIE_Store_R;
output [4:0] TIE_LSSize_R;
output TIE_LSIndexed_R;
output [31:0] TIE_LSOffset_R;
input [127:0] TIE_MemLoadData_M;
output [7:0] TIE_MemStoreData8_E;
output [15:0] TIE_MemStoreData16_E;
output [31:0] TIE_MemStoreData32_E;
output [63:0] TIE_MemStoreData64_E;
output [127:0] TIE_MemStoreData128_E;
output TIE_Stall_R;
output TIE_Exception_E;
output [5:0] TIE_ExcCause_E;
output TIE_bsRead_R;
output TIE_btRead_R;
output TIE_btWrite_R;
output TIE_brWrite_R;
output TIE_bsWrite_R;
output [4:0] TIE_bsReadSize_R;
output [4:0] TIE_btReadSize_R;
output [4:0] TIE_bWriteSize_R;
input [15:0] TIE_bsReadData_E;
input [15:0] TIE_btReadData_E;
output TIE_bWriteData1_E;
output [1:0] TIE_bWriteData2_E;
output [3:0] TIE_bWriteData4_E;
output [7:0] TIE_bWriteData8_E;
output [15:0] TIE_bWriteData16_E;
output TIE_bDataKill_E;
input [7:0] CPEnable;
input [23:0] Instr_R;
input [31:0] SBus_E;
input [31:0] TBus_E;
input [31:0] MemOpAddr_E;
input Kill_E;
input Except_W;
input Replay_W;
input G1WCLK;
```

```
input Reset;

// unused signals
wire TMode = 0;

// control signals
wire KillPipe_W;
wire clk;

// decoded signals
wire GFADD8_C0;
wire GFADD8I_C0;
wire GFMULX8_C0;
wire GFRWMOD8_C0;
wire LGF8_I_C0;
wire SGF8_I_C0;
wire LGF8_IU_C0;
wire SGF8_IU_C0;
wire LGF8_X_C0;
wire SGF8_X_C0;
wire LGF8_XU_C0;
wire SGF8_XU_C0;
wire RUR0_C0;
wire WUR0_C0;
wire [31:0] imm4_C0;
wire [7:0] imm8_C0;
wire art_use_C0;
wire art_def_C0;
wire ars_use_C0;
wire ars_def_C0;
wire arr_use_C0;
wire arr_def_C0;
wire br_use_C0;
wire br_def_C0;
wire bs_use_C0;
wire bs_def_C0;
wire bt_use_C0;
wire bt_def_C0;
wire bs4_use_C0;
wire bs4_def_C0;
wire bs8_use_C0;
wire bs8_def_C0;
wire gr_use_C0;
wire gr_def_C0;
wire gs_use_C0;
wire gs_def_C0;
wire gt_use_C0;
wire gt_def_C0;
wire gfmod_use1_C0;
wire gfmod_def1_C0;
wire AR_rd0_use1_C0;
wire AR_rd0_width32_C0;
wire AR_rd1_use1_C0;
wire AR_rd1_width32_C0;
wire AR_wd_def1_C0;
wire AR_wd_width32_C0;
wire [3:0] gf_rd0_addr_C0;
```

```
wire gf_rd0_use1_C0;
wire gf_rd0_width8_C0;
wire [3:0] gf_rd1_addr_C0;
wire gf_rd1_use1_C0;
wire gf_rd1_width8_C0;
wire [3:0] gf_rd2_addr_C0;
wire gf_rd2_use1_C0;
wire gf_rd2_width8_C0;
wire [3:0] gf_wd_addr_C0;
wire gf_wd_def2_C0;
wire gf_wd_def1_C0;
wire gf_wd_width8_C0;
wire gf1_semantic_C0;
wire gf4_semantic_C0;
wire gf2_semantic_C0;
wire gf3_semantic_C0;
wire lgf_semantic_C0;
wire sgf_semantic_C0;
wire RUR0_semantic_C0;
wire WUR0_semantic_C0;
wire load_instruction_C0;
wire store_instruction_C0;
wire TIE_Inst_C0;
wire [23:0] Inst_C0;

// state data, write-enable and stall signals
wire [7:0] gfmod_ps_C1;
wire [7:0] gfmod_ns_C1;
wire gfmod_kill_C1;
wire gfmod_Stall_C1;

// register data, write-enable and stall signals
wire [31:0] AR_rd0_data_C1;
wire [31:0] AR_rd1_data_C1;
wire [31:0] AR_wd_data32_C1;
wire AR_wd_kill_C1;
wire [7:0] gf_rd0_data_C1;
wire [7:0] gf_rd1_data_C1;
wire [7:0] gf_rd2_data_C1;
wire [7:0] gf_wd_data8_C2;
wire gf_wd_kill_C2;
wire [7:0] gf_wd_data8_C1;
wire gf_wd_kill_C1;
wire gf_Stall_C1;

// operands
wire [31:0] art_i_C1;
wire [31:0] art_o_C1;
wire art_kill_C1;
wire [31:0] ars_i_C1;
wire [31:0] ars_o_C1;
wire ars_kill_C1;
wire [31:0] arr_o_C1;
wire arr_kill_C1;
wire [7:0] gr_i_C1;
wire [7:0] gr_o_C2;
wire gr_kill_C2;
```

```
wire [7:0] gr_o_C1;
wire gr_kill_C1;
wire [7:0] gs_i_C1;
wire [7:0] gt_i_C1;
wire [7:0] gt_o_C2;
wire gt_kill_C2;
wire [7:0] gt_o_C1;
wire gt_kill_C1;

// output state of semantic gf1

// output interface of semantic gf1

// output operand of semantic gf1
wire [7:0] gf1_gr_o_C1;
wire gf1_gr_kill_C1;

// output state of semantic gf4

// output interface of semantic gf4

// output operand of semantic gf4
wire [7:0] gf4_gr_o_C1;
wire gf4_gr_kill_C1;

// output state of semantic gf2

// output interface of semantic gf2

// output operand of semantic gf2
wire [7:0] gf2_gr_o_C1;
wire gf2_gr_kill_C1;

// output state of semantic gf3
wire [7:0] gf3_gfmod_ns_C1;
wire gf3_gfmod_kill_C1;

// output interface of semantic gf3

// output operand of semantic gf3
wire [7:0] gf3_gt_o_C1;
wire gf3_gt_kill_C1;

// output state of semantic lgf

// output interface of semantic lgf
wire [4:0] lgf_LSSize_C0;
wire [31:0] lgf_VAddrBase_C1;
wire [31:0] lgf_VAddrIndex_C1;
wire [31:0] lgf_VAddrOffset_C0;
wire lgf_LSIndexed_C0;

// output operand of semantic lgf
wire [7:0] lgf_gt_o_C2;
wire lgf_gt_kill_C2;
wire [31:0] lgf_ars_o_C1;
wire lgf_ars_kill_C1;
```

```
wire [7:0] lgf_gr_o_C2;
wire lgf_gr_kill_C2;

// output state of semantic sgf

// output interface of semantic sgf
wire [4:0] sgf_LSSize_C0;
wire [7:0] sgf_MemDataOut8_C1;
wire [31:0] sgf_VAddrBase_C1;
wire [31:0] sgf_VAddrIndex_C1;
wire [31:0] sgf_VAddrOffset_C0;
wire sgf_LSIndexed_C0;

// output operand of semantic sgf
wire [31:0] sgf_ars_o_C1;
wire sgf_ars_kill_C1;

// output state of semantic RUR0

// output interface of semantic RUR0

// output operand of semantic RUR0
wire [31:0] RUR0_arr_o_C1;
wire RUR0_arr_kill_C1;

// output state of semantic WUR0
wire [7:0] WUR0_gfmod_ns_C1;
wire WUR0_gfmod_kill_C1;

// output interface of semantic WUR0

// output operand of semantic WUR0

// TIE-defined interface signals
wire [31:0] VAddr_C1;
wire [31:0] VAddrBase_C1;
wire [31:0] VAddrOffset_C0;
wire [31:0] VAddrIndex_C1;
wire [31:0] VAddrIn_C1;
wire [4:0] LSSize_C0;
wire LSIndexed_C0;
wire [127:0] MemDataIn128_C2;
wire [63:0] MemDataIn64_C2;
wire [31:0] MemDataIn32_C2;
wire [15:0] MemDataIn16_C2;
wire [7:0] MemDataIn8_C2;
wire [127:0] MemDataOut128_C1;
wire [63:0] MemDataOut64_C1;
wire [31:0] MemDataOut32_C1;
wire [15:0] MemDataOut16_C1;
wire [7:0] MemDataOut8_C1;
wire Exception_C1;
wire [5:0] ExcCause_C1;
wire [7:0] CPEnable_C1;
    xtflop #(1) reset(localReset, Reset, G1WCLK);

xmTIE_decoder TIE_decoder (
```

```
.GFADD8(GFADD8_C0),
.GFADD8I(GFADD8I_C0),
.GFMULX8(GFMULX8_C0),
.GFRWMOD8(GFRWMOD8_C0),
.LGF8_I(LGF8_I_C0),
.SGF8_I(SGF8_I_C0),
.LGF8_IU(LGF8_IU_C0),
.SGF8_IU(SGF8_IU_C0),
.LGF8_X(LGF8_X_C0),
.SGF8_X(SGF8_X_C0),
.LGF8_XU(LGF8_XU_C0),
.SGF8_XU(SGF8_XU_C0),
.RUR0(RUR0_C0),
.WUR0(WUR0_C0),
.imm4(imm4_C0),
.imm8(imm8_C0),
.art_use(art_use_C0),
.art_def(art_def_C0),
.ars_use(ars_use_C0),
.ars_def(ars_def_C0),
.arr_use(arr_use_C0),
.arr_def(arr_def_C0),
.br_use(br_use_C0),
.br_def(br_def_C0),
.bs_use(bs_use_C0),
.bs_def(bs_def_C0),
.bt_use(bt_use_C0),
.bt_def(bt_def_C0),
.bs4_use(bs4_use_C0),
.bs4_def(bs4_def_C0),
.bs8_use(bs8_use_C0),
.bs8_def(bs8_def_C0),
.gr_use(gr_use_C0),
.gr_def(gr_def_C0),
.gs_use(gs_use_C0),
.gs_def(gs_def_C0),
.gt_use(gt_use_C0),
.gt_def(gt_def_C0),
.gfmod_use1(gfmod_use1_C0),
.gfmod_def1(gfmod_def1_C0),
.AR_rd0_use1(AR_rd0_use1_C0),
.AR_rd0_width32(AR_rd0_width32_C0),
.AR_rd1_use1(AR_rd1_use1_C0),
.AR_rd1_width32(AR_rd1_width32_C0),
.AR_wd_def1(AR_wd_def1_C0),
.AR_wd_width32(AR_wd_width32_C0),
.gf_rd0_addr(gf_rd0_addr_C0),
.gf_rd0_use1(gf_rd0_use1_C0),
.gf_rd0_width8(gf_rd0_width8_C0),
.gf_rd1_addr(gf_rd1_addr_C0),
.gf_rd1_use1(gf_rd1_use1_C0),
.gf_rd1_width8(gf_rd1_width8_C0),
.gf_rd2_addr(gf_rd2_addr_C0),
.gf_rd2_use1(gf_rd2_use1_C0),
.gf_rd2_width8(gf_rd2_width8_C0),
.gf_wd_addr(gf_wd_addr_C0),
.gf_wd_def2(gf_wd_def2_C0),
```

```
    .gf_wd_def1(gf_wd_def1_C0),
    .gf_wd_width8(gf_wd_width8_C0),
    .gf1_semantic(gf1_semantic_C0),
    .gf4_semantic(gf4_semantic_C0),
    .gf2_semantic(gf2_semantic_C0),
    .gf3_semantic(gf3_semantic_C0),
    .lgf_semantic(lgf_semantic_C0),
    .sgf_semantic(sgf_semantic_C0),
    .RUR0_semantic(RUR0_semantic_C0),
    .WUR0_semantic(WUR0_semantic_C0),
    .load_instruction(load_instruction_C0),
    .store_instruction(store_instruction_C0),
    .TIE_Inst(TIE_Inst_C0),
    .Inst(Inst_C0)
);

xmTIE_gf1 TIE_gf1(
    .GFADD8_C0(GFADD8_C0),
    .gr_o_C1(gf1_gr_o_C1),
    .gr_kill_C1(gf1_gr_kill_C1),
    .gs_i_C1(gs_i_C1),
    .gt_i_C1(gt_i_C1),
    .clk(clk));

xmTIE_gf4 TIE_gf4(
    .GFADD8I_C0(GFADD8I_C0),
    .gr_o_C1(gf4_gr_o_C1),
    .gr_kill_C1(gf4_gr_kill_C1),
    .gs_i_C1(gs_i_C1),
    .imm4_C0(imm4_C0),
    .clk(clk));

xmTIE_gf2 TIE_gf2(
    .GFMULX8_C0(GFMULX8_C0),
    .gr_o_C1(gf2_gr_o_C1),
    .gr_kill_C1(gf2_gr_kill_C1),
    .gs_i_C1(gs_i_C1),
    .gfmod_ps_C1(gfmod_ps_C1),
    .clk(clk));

xmTIE_gf3 TIE_gf3(
    .GFRWMOD8_C0(GFRWMOD8_C0),
    .gt_i_C1(gt_i_C1),
    .gt_o_C1(gf3_gt_o_C1),
    .gt_kill_C1(gf3_gt_kill_C1),
    .gfmod_ps_C1(gfmod_ps_C1),
    .gfmod_ns_C1(gf3_gfmod_ns_C1),
    .gfmod_kill_C1(gf3_gfmod_kill_C1),
    .clk(clk));

xmTIE_lgf TIE_lgf(
    .LGF8_I_C0(LGF8_I_C0),
    .LGF8_IU_C0(LGF8_IU_C0),
    .LGF8_X_C0(LGF8_X_C0),
    .LGF8_XU_C0(LGF8_XU_C0),
    .gt_o_C2(lgf_gt_o_C2),
    .gt_kill_C2(lgf_gt_kill_C2),
```

```
        .ars_i_C1(ars_i_C1),
        .ars_o_C1(lgf_ars_o_C1),
        .ars_kill_C1(lgf_ars_kill_C1),
        .imm8_C0(imm8_C0),
        .gr_o_C2(lgf_gr_o_C2),
        .gr_kill_C2(lgf_gr_kill_C2),
        .art_i_C1(art_i_C1),
        .MemDataIn8_C2(MemDataIn8_C2),
        .VAddrIn_C1(VAddrIn_C1),
        .LSSize_C0(lgf_LSSize_C0),
        .VAddrBase_C1(lgf_VAddrBase_C1),
        .VAddrIndex_C1(lgf_VAddrIndex_C1),
        .VAddrOffset_C0(lgf_VAddrOffset_C0),
        .LSIndexed_C0(lgf_LSIndexed_C0),
        .clk(clk));

    xmTIE_sgf TIE_sgf(
        .SGF8_I_C0(SGF8_I_C0),
        .SGF8_IU_C0(SGF8_IU_C0),
        .SGF8_X_C0(SGF8_X_C0),
        .SGF8_XU_C0(SGF8_XU_C0),
        .gt_i_C1(gt_i_C1),
        .ars_i_C1(ars_i_C1),
        .ars_o_C1(sgf_ars_o_C1),
        .ars_kill_C1(sgf_ars_kill_C1),
        .imm8_C0(imm8_C0),
        .gr_i_C1(gr_i_C1),
        .art_i_C1(art_i_C1),
        .VAddrIn_C1(VAddrIn_C1),
        .LSSize_C0(sgf_LSSize_C0),
        .MemDataOut8_C1(sgf_MemDataOut8_C1),
        .VAddrBase_C1(sgf_VAddrBase_C1),
        .VAddrIndex_C1(sgf_VAddrIndex_C1),
        .VAddrOffset_C0(sgf_VAddrOffset_C0),
        .LSIndexed_C0(sgf_LSIndexed_C0),
        .clk(clk));

    xmTIE_RUR0 TIE_RUR0(
        .RUR0_C0(RUR0_C0),
        .arr_o_C1(RUR0_arr_o_C1),
        .arr_kill_C1(RUR0_arr_kill_C1),
        .gfmod_ps_C1(gfmod_ps_C1),
        .clk(clk));

    xmTIE_WUR0 TIE_WUR0(
        .WUR0_C0(WUR0_C0),
        .art_i_C1(art_i_C1),
        .gfmod_ns_C1(WUR0_gfmod_ns_C1),
        .gfmod_kill_C1(WUR0_gfmod_kill_C1),
        .clk(clk));

    xmTIE_gfmod_State TIE_gfmod_State (
        .ps_width8_C0(1'b1),
        .ps_use1_C0(gfmod_use1_C0),
        .ps_data_C1(gfmod_ps_C1),
        .ns_width8_C0(1'b1),
        .ns_def1_C0(gfmod_def1_C0),
```

```
        .ns_data8_C1(gfmod_ns_C1),
        .ns_wen_C1(~gfmod_kill_C1),
        .Kill_E(Kill_E),
        .KillPipe_W(KillPipe_W),
        .Stall_R(gfmod_Stall_C1),
        .clk(clk)
    );

    xmTIE_gf_Regfile TIE_gf_Regfile (
        .rd0_addr_C0(gf_rd0_addr_C0),
        .rd0_use1_C0(gf_rd0_use1_C0),
        .rd0_data_C1(gf_rd0_data_C1),
        .rd0_width8_C0(gf_rd0_width8_C0),
        .rd1_addr_C0(gf_rd1_addr_C0),
        .rd1_use1_C0(gf_rd1_use1_C0),
        .rd1_data_C1(gf_rd1_data_C1),
        .rd1_width8_C0(gf_rd1_width8_C0),
        .rd2_addr_C0(gf_rd2_addr_C0),
        .rd2_use1_C0(gf_rd2_use1_C0),
        .rd2_data_C1(gf_rd2_data_C1),
        .rd2_width8_C0(gf_rd2_width8_C0),
        .wd_addr_C0(gf_wd_addr_C0),
        .wd_def2_C0(gf_wd_def2_C0),
        .wd_wen_C2(~gf_wd_kill_C2),
        .wd_data8_C2(gf_wd_data8_C2),
        .wd_def1_C0(gf_wd_def1_C0),
        .wd_wen_C1(~gf_wd_kill_C1),
        .wd_data8_C1(gf_wd_data8_C1),
        .wd_width8_C0(gf_wd_width8_C0),
        .Kill_E(Kill_E),
        .KillPipe_W(KillPipe_W),
        .Stall_R(gf_Stall_C1),
        .clk(clk)
    );

    // Stall logic
    assign TIE_Stall_R = 1'b0
              | gf_Stall_C1
              | gfmod_Stall_C1;

    // pipeline semantic select signals to each stage
    wire lgf_semantic_C1;
    xtdelay1 #(1) ilgf_semantic_C1(.xtin(lgf_semantic_C0), .xtout(lgf_semantic_C1),
    .clk(clk));
    wire sgf_semantic_C1;
    xtdelay1 #(1) isgf_semantic_C1(.xtin(sgf_semantic_C0), .xtout(sgf_semantic_C1),
    .clk(clk));
    wire gf3_semantic_C1;
    xtdelay1 #(1) igf3_semantic_C1(.xtin(gf3_semantic_C0), .xtout(gf3_semantic_C1),
    .clk(clk));
    wire WUR0_semantic_C1;
    xtdelay1 #(1) iWUR0_semantic_C1(.xtin(WUR0_semantic_C0),
    .xtout(WUR0_semantic_C1), .clk(clk));
    wire RUR0_semantic_C1;
    xtdelay1 #(1) iRUR0_semantic_C1(.xtin(RUR0_semantic_C0),
    .xtout(RUR0_semantic_C1), .clk(clk));
    wire lgf_semantic_C2;
```

```
xtdelay2 #(1) ilgf_semantic_C2(.xtin(lgf_semantic_C0), .xtout(lgf_semantic_C2),
.clk(clk));
wire gf1_semantic_C1;
xtdelay1 #(1) igf1_semantic_C1(.xtin(gf1_semantic_C0), .xtout(gf1_semantic_C1),
.clk(clk));
wire gf4_semantic_C1;
xtdelay1 #(1) igf4_semantic_C1(.xtin(gf4_semantic_C0), .xtout(gf4_semantic_C1),
.clk(clk));
wire gf2_semantic_C1;
xtdelay1 #(1) igf2_semantic_C1(.xtin(gf2_semantic_C0), .xtout(gf2_semantic_C1),
.clk(clk));

// combine output interface signals from all semantics
assign VAddr_C1 = 32'b0;
assign VAddrBase_C1 = 32'b0
          | (lgf_VAddrBase_C1 & {32{lgf_semantic_C1}})
          | (sgf_VAddrBase_C1 & {32{sgf_semantic_C1}});
assign VAddrOffset_C0 = 32'b0
          | (lgf_VAddrOffset_C0 & {32{lgf_semantic_C0}})
          | (sgf_VAddrOffset_C0 & {32{sgf_semantic_C0}});
assign VAddrIndex_C1 = 32'b0
          | (lgf_VAddrIndex_C1 & {32{lgf_semantic_C1}})
          | (sgf_VAddrIndex_C1 & {32{sgf_semantic_C1}});
assign LSSize_C0 = 5'b0
          | (lgf_LSSize_C0 & {5{lgf_semantic_C0}})
          | (sgf_LSSize_C0 & {5{sgf_semantic_C0}});
assign LSIndexed_C0 = 1'b0
          | (lgf_LSIndexed_C0 & lgf_semantic_C0)
          | (sgf_LSIndexed_C0 & sgf_semantic_C0);
assign MemDataOut128_C1 = 128'b0;
assign MemDataOut64_C1 = 64'b0;
assign MemDataOut32_C1 = 32'b0;
assign MemDataOut16_C1 = 16'b0;
assign MemDataOut8_C1 = 8'b0
          | (sgf_MemDataOut8_C1 & {8{sgf_semantic_C1}});
assign Exception_C1 = 1'b0;
assign ExcCause_C1 = 6'b0;

// combine output state signals from all semantics
assign gfmod_ns_C1 = 8'b0
          | (gf3_gfmod_ns_C1 & {8{gf3_semantic_C1}})
          | (WUR0_gfmod_ns_C1 & {8{WUR0_semantic_C1}});
assign gfmod_kill_C1 = 1'b0
          | (gf3_gfmod_kill_C1 & gf3_semantic_C1)
          | (WUR0_gfmod_kill_C1 & WUR0_semantic_C1);

// combine output operand signals from all semantics
assign art_o_C1 = 32'b0;
assign art_kill_C1 = 1'b0;
assign ars_o_C1 = 32'b0
          | (lgf_ars_o_C1 & {32{lgf_semantic_C1}})
          | (sgf_ars_o_C1 & {32{sgf_semantic_C1}});
assign ars_kill_C1 = 1'b0
          | (lgf_ars_kill_C1 & lgf_semantic_C1)
          | (sgf_ars_kill_C1 & sgf_semantic_C1);
assign arr_o_C1 = 32'b0
          | (RUR0_arr_o_C1 & {32{RUR0_semantic_C1}});
```

```
assign arr_kill_C1 = 1'b0
          | (RUR0_arr_kill_C1 & RUR0_semantic_C1);
assign gr_o_C2 = 8'b0
          | (lgf_gr_o_C2 & {8{lgf_semantic_C2}});
assign gr_kill_C2 = 1'b0
          | (lgf_gr_kill_C2 & lgf_semantic_C2);
assign gr_o_C1 = 8'b0
          | (gf1_gr_o_C1 & {8{gf1_semantic_C1}})
          | (gf4_gr_o_C1 & {8{gf4_semantic_C1}})
          | (gf2_gr_o_C1 & {8{gf2_semantic_C1}});
assign gr_kill_C1 = 1'b0
          | (gf1_gr_kill_C1 & gf1_semantic_C1)
          | (gf4_gr_kill_C1 & gf4_semantic_C1)
          | (gf2_gr_kill_C1 & gf2_semantic_C1);
assign gt_o_C2 = 8'b0
          | (lgf_gt_o_C2 & {8{lgf_semantic_C2}});
assign gt_kill_C2 = 1'b0
          | (lgf_gt_kill_C2 & lgf_semantic_C2);
assign gt_o_C1 = 8'b0
          | (gf3_gt_o_C1 & {8{gf3_semantic_C1}});
assign gt_kill_C1 = 1'b0
          | (gf3_gt_kill_C1 & gf3_semantic_C1);

// output operand to write port mapping logic
assign AR_wd_data32_C1 = ars_o_C1 | arr_o_C1 | 32'b0;
assign AR_wd_kill_C1 = ars_kill_C1 | arr_kill_C1 | 1'b0;
assign gf_wd_data8_C2 = gt_o_C2 | gr_o_C2 | 8'b0;
assign gf_wd_kill_C2 = gt_kill_C2 | gr_kill_C2 | 1'b0;
assign gf_wd_data8_C1 = gr_o_C1 | gt_o_C1 | 8'b0;
assign gf_wd_kill_C1 = gr_kill_C1 | gt_kill_C1 | 1'b0;

// read port to input operand mapping logic
assign ars_i_C1 = AR_rd0_data_C1;
assign art_i_C1 = AR_rd1_data_C1;
assign gs_i_C1 = gf_rd0_data_C1;
assign gt_i_C1 = gf_rd1_data_C1;
assign gr_i_C1 = gf_rd2_data_C1;

// clock and instructions
assign clk = G1WCLK;
assign Inst_C0 = Instr_R;
assign TIE_inst_R = TIE_Inst_C0;

// AR-related signals to/from core
assign TIE_asRead_R = ars_use_C0;
assign TIE_atRead_R = art_use_C0;
assign TIE_atWrite_R = art_def_C0;
assign TIE_arWrite_R = arr_def_C0;
assign TIE_asWrite_R = ars_def_C0;
assign TIE_aWriteM_R = 0;
assign TIE_aWriteData_E = AR_wd_data32_C1;
assign TIE_aWriteData_M = 0;
assign TIE_aDataKill_E = AR_wd_kill_C1;
assign TIE_aDataKill_M = 0;
assign AR_rd0_data_C1 = SBus_E;
assign AR_rd1_data_C1 = TBus_E;
```

```
// BR-related signals to/from core
assign TIE_bsRead_R = 1'b0 | bs_use_C0 | bs4_use_C0 | bs8_use_C0;
assign TIE_btRead_R = 1'b0 | bt_use_C0;
assign TIE_btWrite_R = 1'b0 | bt_def_C0;
assign TIE_bsWrite_R = 1'b0 | bs_def_C0 | bs4_def_C0 | bs8_def_C0;
assign TIE_brWrite_R = 1'b0 | br_def_C0;
assign TIE_bWriteData16_E = 0;
assign TIE_bWriteData8_E = 0;
assign TIE_bWriteData4_E = 0;
assign TIE_bWriteData2_E = 0;
assign TIE_bWriteData1_E = 0;
assign TIE_bDataKill_E = 0;
assign TIE_bWriteSize_R = {1'b0, 1'b0, 1'b0, 1'b0, 1'b0};
assign TIE_bsReadSize_R = {1'b0, 1'b0, 1'b0, 1'b0, 1'b0};
assign TIE_btReadSize_R = {1'b0, 1'b0, 1'b0, 1'b0, 1'b0};

// Load/store signals to/from core
assign TIE_Load_R = load_instruction_C0;
assign TIE_Store_R = store_instruction_C0;
assign TIE_LSSize_R = LSSize_C0;
assign TIE_LSIndexed_R = LSIndexed_C0;
assign TIE_LSOffset_R = VAddrOffset_C0;
assign TIE_MemStoreData128_E = MemDataOut128_C1;
assign TIE_MemStoreData64_E = MemDataOut64_C1;
assign TIE_MemStoreData32_E = MemDataOut32_C1;
assign TIE_MemStoreData16_E = MemDataOut16_C1;
assign TIE_MemStoreData8_E = MemDataOut8_C1;
assign MemDataIn128_C2 = TIE_MemLoadData_M;
assign MemDataIn64_C2 = TIE_MemLoadData_M;
assign MemDataIn32_C2 = TIE_MemLoadData_M;
assign MemDataIn16_C2 = TIE_MemLoadData_M;
assign MemDataIn8_C2 = TIE_MemLoadData_M;
assign VAddrIn_C1 = MemOpAddr_E;

// CPEnable and control signals to/from core
assign CPEnable_C1 = CPEnable;
assign TIE_Exception_E = Exception_C1;
assign TIE_ExcCause_E = ExcCause_C1;
assign KillPipe_W = Except_W | Replay_W;
endmodule
module xtdelay1(xtout, xtin, clk);
parameter size = 1;
output [size-1:0] xtout;
input [size-1:0] xtin;
input clk;
    wire [size-1:0] t0;
    xtflop #(size) i0(t0, xtin, clk);
    assign xtout = t0;
endmodule

module xtdelay2(xtout, xtin, clk);
parameter size = 1;
output [size-1:0] xtout;
input [size-1:0] xtin;
input clk;
    wire [size-1:0] t0;
    xtflop #(size) i0(t0, xtin, clk);
```

```
    wire [size-1:0] t1;
    xtflop #(size) i1(t1, t0, clk);
    assign xtout = t1;
endmodule

module xtmux3p(o, d0, d1, d2, s0, s1);
parameter size = 1;
output [size-1:0] o;
input [size-1:0] d0, d1, d2;
input s0, s1;
    wire [1:0] s = s0 ? 0 : s1 ? 1 : 2;
    xtmux3e #(size) i0(o, d0, d1, d2, s);
endmodule

module xtregfile_1R1W_1(rd0_data, wr0_data, wr0_we, clk);
parameter size=32, addr_size=0;
output [size-1:0] rd0_data;
input [size-1:0] wr0_data;
input wr0_we;
input clk;
    wire wr0_addr = 0;

    wire word0_we = wr0_we & (wr0_addr == 0);
    wire [size-1:0] word0;
    xtenflop #(size) iword0(word0, wr0_data, word0_we, clk);

    assign rd0_data = word0;
endmodule

module xtregfile_3R1W_16(rd0_data, rd0_addr, rd1_data, rd1_addr, rd2_data,
  rd2_addr, wr0_data, wr0_addr, wr0_we, clk);
parameter size=32, addr_size=4;
output [size-1:0] rd0_data;
input [addr_size-1:0] rd0_addr;
output [size-1:0] rd1_data;
input [addr_size-1:0] rd1_addr;
output [size-1:0] rd2_data;
input [addr_size-1:0] rd2_addr;
input [size-1:0] wr0_data;
input [addr_size-1:0] wr0_addr;
input wr0_we;
input clk;
    wire [size-1:0] wr0_ndata;
    xtnflop #(size) iwr0_ndata(wr0_ndata, wr0_data, clk);

    wire word0_we = wr0_we & (wr0_addr == 0);
    wire [size-1:0] word0;
    wire gclk0;
    xtclock_gate_nor xt_clock_gate_nor0(gclk0, clk, ~word0_we);
    xtRFlatch #(size) iword0(word0, wr0_ndata, gclk0);

    wire word1_we = wr0_we & (wr0_addr == 1);
    wire [size-1:0] word1;
    wire gclk1;
    xtclock_gate_nor xt_clock_gate_nor1(gclk1, clk, ~word1_we);
    xtRFlatch #(size) iword1(word1, wr0_ndata, gclk1);
```

Versifi deal specifics

Per our conversation, below are the deal specifics with regard to Versifi.

- PurchasePro.com will pay $20 million for an equity stake in Versifi. The components of the $20 million investment consist of $10 million for a perpetual license to use Versifi's software product and $10 in PurchasePro.com series preferred stock.
- Concurrently PurchasePro.com will pay $4 million for $7 million worth of integration services from Capita (discounted contract) for the completion of all 20 Advanstar sites.
- In addition, a revenue sharing agreement will be formed, whereby PurchasePro.com will receive 10% of the first $30 million in revenue generated by PPRO for Capita. Thereafter, PPRO will receive 20% of the revenue generated for Capita.

PurchasePro.com series preferred will include the following:

- Blocking right on sale
- Covenant stating that Versifi will not have the right to sell to competitor until after IPO
- PPRO will maintain 1 board seat (7 total seats; 4 outside members)

```
wire word2_we = wr0_we & (wr0_addr == 2);
wire [size-1:0] word2;
wire gclk2;
xtclock_gate_nor xt_clock_gate_nor2(gclk2, clk, ~word2_we);
xtRFlatch #(size) iword2(word2, wr0_ndata, gclk2);

wire word3_we = wr0_we & (wr0_addr == 3);
wire [size-1:0] word3;
wire gclk3;
xtclock_gate_nor xt_clock_gate_nor3(gclk3, clk, ~word3_we);
xtRFlatch #(size) iword3(word3, wr0_ndata, gclk3);

wire word4_we = wr0_we & (wr0_addr == 4);
wire [size-1:0] word4;
wire gclk4;
xtclock_gate_nor xt_clock_gate_nor4(gclk4, clk, ~word4_we);
xtRFlatch #(size) iword4(word4, wr0_ndata, gclk4);

wire word5_we = wr0_we & (wr0_addr == 5);
wire [size-1:0] word5;
wire gclk5;
xtclock_gate_nor xt_clock_gate_nor5(gclk5, clk, ~word5_we);
xtRFlatch #(size) iword5(word5, wr0_ndata, gclk5);

wire word6_we = wr0_we & (wr0_addr == 6);
wire [size-1:0] word6;
wire gclk6;
xtclock_gate_nor xt_clock_gate_nor6(gclk6, clk, ~word6_we);
xtRFlatch #(size) iword6(word6, wr0_ndata, gclk6);

wire word7_we = wr0_we & (wr0_addr == 7);
wire [size-1:0] word7;
wire gclk7;
xtclock_gate_nor xt_clock_gate_nor7(gclk7, clk, ~word7_we);
xtRFlatch #(size) iword7(word7, wr0_ndata, gclk7);

wire word8_we = wr0_we & (wr0_addr == 8);
wire [size-1:0] word8;
wire gclk8;
xtclock_gate_nor xt_clock_gate_nor8(gclk8, clk, ~word8_we);
xtRFlatch #(size) iword8(word8, wr0_ndata, gclk8);

wire word9_we = wr0_we & (wr0_addr == 9);
wire [size-1:0] word9;
wire gclk9;
xtclock_gate_nor xt_clock_gate_nor9(gclk9, clk, ~word9_we);
xtRFlatch #(size) iword9(word9, wr0_ndata, gclk9);

wire word10_we = wr0_we & (wr0_addr == 10);
wire [size-1:0] word10;
wire gclk10;
xtclock_gate_nor xt_clock_gate_nor10(gclk10, clk, ~word10_we);
xtRFlatch #(size) iword10(word10, wr0_ndata, gclk10);

wire word11_we = wr0_we & (wr0_addr == 11);
wire [size-1:0] word11;
wire gclk11;
```

```
    xtclock_gate_nor xt_clock_gate_nor11(gclk11, clk, ~word11_we);
    xtRFlatch #(size) iword11(word11, wr0_ndata, gclk11);

    wire word12_we = wr0_we & (wr0_addr == 12);
    wire [size-1:0] word12;
    wire gclk12;
    xtclock_gate_nor xt_clock_gate_nor12(gclk12, clk, ~word12_we);
    xtRFlatch #(size) iword12(word12, wr0_ndata, gclk12);

    wire word13_we = wr0_we & (wr0_addr == 13);
    wire [size-1:0] word13;
    wire gclk13;
    xtclock_gate_nor xt_clock_gate_nor13(gclk13, clk, ~word13_we);
    xtRFlatch #(size) iword13(word13, wr0_ndata, gclk13);

    wire word14_we = wr0_we & (wr0_addr == 14);
    wire [size-1:0] word14;
    wire gclk14;
    xtclock_gate_nor xt_clock_gate_nor14(gclk14, clk, ~word14_we);
    xtRFlatch #(size) iword14(word14, wr0_ndata, gclk14);

    wire word15_we = wr0_we & (wr0_addr == 15);
    wire [size-1:0] word15;
    wire gclk15;
    xtclock_gate_nor xt_clock_gate_nor15(gclk15, clk, ~word15_we);
    xtRFlatch #(size) iword15(word15, wr0_ndata, gclk15);

    xtmux16e #(size) rd0(rd0_data, word0, word1, word2, word3, word4, word5,
word6, word7, word8, word9, word10, word11, word12, word13, word14, word15,
rd0_addr);
    xtmux16e #(size) rd1(rd1_data, word0, word1, word2, word3, word4, word5,
word6, word7, word8, word9, word10, word11, word12, word13, word14, word15,
rd1_addr);
    xtmux16e #(size) rd2(rd2_data, word0, word1, word2, word3, word4, word5,
word6, word7, word8, word9, word10, word11, word12, word13, word14, word15,
rd2_addr);
endmodule

module xtmux16e(o, d0, d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12, d13,
d14, d15, s);
parameter size = 1;
output [size-1:0] o;
input [size-1:0] d0, d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12, d13,
d14, d15;
input [3:0] s;
    wire [size-1:0] t0;
    xtmux4e #(size) i0(t0, d0, d1, d2, d3, {s[1], s[0]});
    wire [size-1:0] t1;
    xtmux4e #(size) i1(t1, d4, d5, d6, d7, {s[1], s[0]});
    wire [size-1:0] t2;
    xtmux4e #(size) i2(t2, d8, d9, d10, d11, {s[1], s[0]});
    wire [size-1:0] t3;
    xtmux4e #(size) i3(t3, d12, d13, d14, d15, {s[1], s[0]});
    wire [size-1:0] t4;
    xtmux4e #(size) i4(t4, t0, t1, t2, t3, {s[3], s[2]});
    assign o = t4;
endmodule
```

```
module xtRFenlatch(xtRFenlatchout,xtin,xten,clk);
   parameter size = 32;
   output [size-1:0] xtRFenlatchout;
   input [size-1:0]  xtin;
   input          xten;
   input          clk;

   reg [size-1:0]    xtRFenlatchout;

 always @(clk or xten or xtin or xtRFenlatchout) begin
  if (clk) begin
     xtRFenlatchout <= #1 (xten) ? xtin : xtRFenlatchout;
  end
 end

endmodule
module xtRFlatch(xtRFlatchout,xtin,clk);
   parameter size = 32;
   output [size-1:0] xtRFlatchout;
   input [size-1:0]  xtin;
   input          clk;

   reg [size-1:0]    xtRFlatchout;

 always @(clk or xtin) begin
  if (clk) begin
   xtRFlatchout <= #1 xtin;
  end
 end

endmodule
module xtadd(xtout, a, b);
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  a;
   input [size-1:0]  b;

   assign xtout = a + b;

endmodule
module xtaddc(sum, carry, a, b, c);
   parameter size = 32;

   output [size-1:0] sum;
   output         carry;
   input [size-1:0]  a;
   input [size-1:0]  b;
   input          c;

   wire           junk;

   assign {carry, sum, junk} = {a,c} + {b,c};

endmodule
module xtaddcin(xtout, a, b, c);
```

```
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  a;
   input [size-1:0]  b;
   input             c;

   assign xtout = ({a,c} + {b,c}) >> 1;

endmodule
module xtaddcout(sum, carry, a, b);
   parameter size = 1;

   output [size-1:0] sum;
   output            carry;
   input [size-1:0]  a;
   input [size-1:0]  b;

   assign {carry, sum} = a + b;

endmodule
module xtbooth(out, cin, a, b, sign, negate);
parameter size = 16;
output [size+1:0] out;
output cin;
input [size-1:0] a;
input [2:0] b;
input sign, negate;
    wire ase = sign & a[size-1];
    wire [size+1:0] ax1 = {ase, ase, a};
    wire [size+1:0] ax2 = {ase, a, 1'd0};
    wire one = b[1] ^ b[0];
    wire two = b[2] ? ~b[1] & ~b[0] : b[1] & b[0];
    wire cin = negate ? (~b[2] & (b[1] | b[0])) : (b[2] & ~(b[1] & b[0]));
    assign out = {size+2{cin}} ^ (ax1&{size+2{one}} | ax2&{size+2{two}});
endmodule
module xtclock_gate_nor(xtout,xtin1,xtin2);
 output xtout;
 input xtin1,xtin2;

 assign xtout = ~(xtin1 || xtin2);

endmodule
module xtclock_gate_or(xtout,xtin1,xtin2);
 output xtout;
 input xtin1,xtin2;

 assign xtout = (xtin1 || xtin2);

endmodule
module xtcsa (sum, carry, a, b, c);
   parameter size = 1;

   output [size-1:0]    sum;
   output [size-1:0]    carry;
   input [size-1:0]     a;
   input [size-1:0]     b;
```

```
    input [size-1:0]     c;

    assign sum = a ^ b ^ c;
    assign carry = (a & b) | (b & c) | (c & a) ;

endmodule
module xtenflop(xtout, xtin, en, clk);
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  xtin;
   input         en;
   input         clk;
   reg [size-1:0]    tmp;

   assign xtout = tmp;
   always @(posedge clk) begin
      if (en)
      tmp <= #1 xtin;
   end

endmodule
module xtfa(sum, carry, a, b, c);
output sum, carry;
input a, b, c;
    assign sum = a ^ b ^ c;
    assign carry = a & b | a & c | b & c;
endmodule
module xtflop(xtout, xtin, clk);
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  xtin;
   input         clk;
   reg [size-1:0]    tmp;

   assign xtout = tmp;
   always @(posedge clk) begin
      tmp <= #1 xtin;
   end

endmodule
module xtha(sum, carry, a, b);
output sum, carry;
input a, b;
    assign sum = a ^ b;
    assign carry = a & b;
endmodule
module xtinc(xtout, a);
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  a;

   assign xtout = a + 1;

endmodule
```

```
module xtmux2e(xtout, a, b, sel);
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  a;
   input [size-1:0]  b;
   input             sel;

   assign xtout = (~sel) ? a : b;

endmodule
module xtmux3e(xtout, a, b, c, sel);
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  a;
   input [size-1:0]  b;
   input [size-1:0]  c;
   input [1:0]          sel;
   reg [size-1:0]    xtout;


   always @(a or b or c or sel) begin
      xtout = sel[1] ? c : (sel[0] ? b : a);
   end
endmodule
module xtmux4e(xtout, a, b, c, d, sel);
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  a;
   input [size-1:0]  b;
   input [size-1:0]  c;
   input [size-1:0]  d;
   input [1:0]          sel;
   reg [size-1:0]    xtout;

   // synopsys infer_mux "xtmux4e"
   always @(sel or a or b or c or d) begin : xtmux4e
      case (sel)         // synopsys parallel_case full_case
      2'b00:
        xtout = a;
      2'b01:
        xtout = b;
      2'b10:
        xtout = c;
      2'b11:
        xtout = d;
      default:
        xtout = {size{1'bx}};
      endcase // case(sel)
   end // always @ (sel or a or b or c or d)

endmodule
module xtnflop(xtout, xtin, clk);
   parameter size = 32;
```

```
    output [size-1:0] xtout;
    input [size-1:0]  xtin;
    input         clk;
    reg [size-1:0]    tmp;

    assign xtout = tmp;
    always @(negedge clk) begin
       tmp <= #1 xtin;
    end // always @ (negedge clk)

 endmodule
 module xtscflop(xtout, xtin, clrb, clk);  // sync clear ff
    parameter size = 32;

    output [size-1:0] xtout;
    input [size-1:0]  xtin;
    input         clrb;
    input         clk;
    reg [size-1:0]    tmp;

    assign xtout = tmp;
    always @(posedge clk) begin
       if (!clrb) tmp <=  0;
       else tmp <= #1 xtin;
    end

 endmodule
 module xtscenflop(xtout, xtin, en, clrb, clk);  // sync clear
    parameter size = 32;

    output [size-1:0] xtout;
    input [size-1:0]  xtin;
    input         en;
    input             clrb;
    input         clk;
    reg [size-1:0]    tmp;

    assign xtout = tmp;
    always @(posedge clk) begin
       if (!clrb) tmp <= 0;
       else if (en)
       tmp <= #1 xtin;
    end

 endmodule
```

gf_check.dcsh

```
/*
 * Copyright 1999-2000 Tensilica Inc.
 * These coded instructions, statements, and computer programs are
 * Confidential Proprietary Information of Tensilica Inc. and may not be
 * disclosed to third parties or copied in any form, in whole or in part,
 * without the prior written consent of Tensilica Inc.
 */
```

```
/*============================================================================
    Generic setup
 ============================================================================*/
hdlin_auto_save_templates = true
define_design_lib WORK -path workdir
define_name_rules no_slash -restrict "/" -replacement_char "_"
verilogout_no_tri = true
verbose_messages = false
sh mkdir -p workdir
sh date
sh hostname


/*============================================================================
    Read and elaborate the design
 ============================================================================*/
/*
foreach(F, {"gf.v", "gf_FF.v", "gf_tlt.v"}) {
*/
foreach(F, {"gf.v"}) {
    read -f verilog "/home/earl/tensilica/test/gf/gf.out/" + F

    /*
    remove_design find(design, "xtha") >/dev/null
    remove_design find(design, "xtfa") >/dev/null
    remove_design find(design, "xtmux4b") >/dev/null
    read -f verilog "/home/earl/tensilica/test/gf/gf.out/prim.v"
    */

    /*
    elaborate xmTIE
    */

    current_design xmTIE
    link
    ungroup -all -flatten
    check_design

    remove_design find(design, "*")
}

quit
```

gf.dcsh

```
/*
 * Copyright 1999-2000 Tensilica Inc.
 * These coded instructions, statements, and computer programs are
 * Confidential Proprietary Information of Tensilica Inc. and may not be
 * disclosed to third parties or copied in any form, in whole or in part,
 * without the prior written consent of Tensilica Inc.
 */

/*============================================================================
    Generic setup
 ============================================================================*/
```

```
hdlin_auto_save_templates = true
define_design_lib WORK -path workdir
define_name_rules no_slash -restrict "/" -replacement_char "_"
verilogout_no_tri = true
verbose_messages = false
sh mkdir -p workdir
sh date
sh hostname


/*=============================================================================
    Library-specific parameters

    Most are self-explanatory.  Examples for each are shown.

    LIB_MAP_ONLY is a set of gates to use the "set_map_only" attribute for
    Design Compiler.  Typically this should be all 3:1 and 4:1 muxes and
    all half-adders and full-adders.

    LIB_SCAN_FLOP is a set of flops to not use for sequential mapping because
    they represent scan flops in the library.

    LIB_DONT_USE can select target gates in the library not to use.
 ============================================================================*/
synthetic_library = {standard.sldb}
search_path = search_path + {"/cad/artisan/Phantom/synopsys/acb872"}
target_library = slow.db
link_library = {"*"} + target_library + synthetic_library
CLOCK_PERIOD = 6.67            /* target clock period */
CLOCK_SKEW = .35        /* estimated clock skew */
CRITICAL_RANGE = .8            /* keep paths off-critical paths tight */
BOUNDARY_LOAD = slow/INVX1/A  /* typical load */
DRIVE_CELL = DFFX4             /* typical drive cell name */
DRIVE_PIN = Q                  /* typical drive pin name */
DRIVE_PIN_FROM = CK            /* typical drive from pin name */
OPERATING_CONDITION = slow     /* operating conditions */
WIRE_LOAD = TSMC32K_Aggresive /* wire-load model */
LIB_MAP_ONLY = {slow/MX4*, slow/MXI4*, slow/ADDF*, slow/ADDH*}
LIB_SCAN_FLOP = {slow/SDFF*, slow/SEDFF*}
LIB_DONT_USE = {slow/ADDFX4*} + LIB_SCAN_FLOP


/*=============================================================================
    Design-specific parameters

    TIE_DESIGN is the name of the top-level design for optimization.  Typically
    it is "xmTIE" the root of the TIE logic.  However, it can be set to any
    lower-level design (e.g., any single semantic block such as xmTIE_myblock)
    to optimize just that semantic block logic.

    TIE_RETIME enables "optimize_registers" for retiming a TIE pipelined
    design.  It can be set to 0, 1 or 2.  If 0, no retiming is done.  If 1,
    retiming of semantic block logic is done.  If 2, a more aggressive retiming
    is done which includes the control and bypass logic in the register files.
    Retiming requires a Design Compiler Ultra license.

    TIE_MAP_EFFORT controls the Design Compiler effort level on the final pass
```

```
    of incremental compiles.

    AREA_IS_PRIORITY tweaks the optimization script to try for a minimum area
    design.  Use it only when timing constraints are very loose.
 ==========================================================================*/
TIE_DESIGN = xmTIE
TIE_RETIME = 0
TIE_MAP_EFFORT = medium
AREA_IS_PRIORITY = 0


/*==========================================================================
    Configure the synthetic library
 ==========================================================================*/
read standard.sldb
set_dont_use standard.sldb/*/rpl
remove_attribute standard.sldb/*cmp*/rpl dont_use


/*==========================================================================
    Read and elaborate the design
 ==========================================================================*/
read -f verilog "/home/earl/tensilica/test/gf/gf.out/gf.v"
remove_design find(design, "xtha") >/dev/null
remove_design find(design, "xtfa") >/dev/null
remove_design find(design, "xtmux4b") >/dev/null
read -f verilog "/home/earl/tensilica/test/gf/gf.out/prim.v"
elaborate TIE_DESIGN
current_design TIE_DESIGN
link


/*==========================================================================
    Optimize
 ==========================================================================*/
/*
 +------------------------------------------------------------------+
 |         Copyright (c) 1997-2000 Tensilica Inc.                   |
 |                                                                  |
 |   These coded instructions, statements, and computer programs   |
 |   are Confidential Proprietary Information of Tensilica Inc.    |
 |   and may not be disclosed to third parties or copied in any    |
 |   form, in whole or in part, without the prior written          |
 |   consent of Tensilica Inc.                                      |
 +------------------------------------------------------------------+

  Title:    Synthesis script for Tensilica primitives

  Created:  Fri Nov 12, 1999

;#  Author:        Richard Rudell
;#          <rudell@tensilica.com>

  Description:

    The Design Compiler "current_design" is relevant when this script is run.
    A hierarchical search from the current design finds the set of primitives.
```

```
    TENSILICA_SOURCE/Hardware/scripts/syn/Xtensa_cons_generic.dc sets the
    constraints on the primitives.

    The primitives are ungrouped when they are optimized.  Most primitives are
    optimized with a CLOCK_PERIOD of 0 and a CLOCK_SKEW of 0 (i.e., min-
    delay).  Some are mapped with the real constraints.  Not all primitives are
    optimized.

    The primitives are ordered so that primitives which contain other
primitives
    as instances will be optimized later in the flow.  The order is hardwired.

    XTADD and XTMUL give better results when mapped "incremental".  A primitive
    with lots of generic logic when it is mapped usually is worse when mapped
    incremental.

    prim.v contains special synthesis versions of xtmux3e, xtmux4e, and xtcsa.
    These designs contain cells of xtmux3e_1024, xtmux4e_1024, and xtcsa_1024
    which then instantiate 1,024 xtmux3b, xtmux4b, and xtfa cells.  It is
    important that these designs are ungrouped and optimized to remove the many
    nets with no fanout.  This trick is used to ensure efficient cells from the
    library are used, regardless of the width of the primitive.

    Single-bit versions of xtmux3b, xtmux4b, xtfa and xtha are premapped hoping
    to get single cells from the library of they exist.  Note that this is
    pretty much guaranteed for xtmux4b, xtfa, and xtha as they are instantiated
    in "prim.v" as GTECH components.

  Revision History:
      Nov 1999: Rewrite to specialize it for some primitives

      Nov 1998: Original version
*/


XTVERBOSE = 0
XTCURRENT_DESIGN = current_design
XTCLOCK_PERIOD = CLOCK_PERIOD
XTCLOCK_SKEW = CLOCK_SKEW
LAST_TIME = time()

/* configure the library */
read target_library
set_map_only LIB_MAP_ONLY + {gtech/GTECH_ADD_ABC, gtech/GTECH_ADD_AB,
gtech/GTECH_MUX4} true
if (LIB_DONT_USE != {}) {
    set_dont_use LIB_DONT_USE
}

current_design XTCURRENT_DESIGN
XTGATE = find(design, "xtmux*b", -hier) + find(design, "xtfa", -hier) +
find(design, "xtha", -hier) >/dev/null
XTCLOCKGATE = find(design, "xtclock_gate*", -hier) >/dev/null
XTRFLATCH = find(design, "xtRF*latch*", -hier) >/dev/null
XTMUX2 = find(design, "xtmux2_size*", -hier) + find(design, "xtmux2e_size*", -
hier) + find(design, "xtmux2p_size*", -hier) >/dev/null
```

```
XTMUX3 = find(design, "xtmux3_size*", -hier) + find(design, "xtmux3e_size*", -
hier) + find(design, "xtmux3p_size*", -hier) >/dev/null
XTMUX4 = find(design, "xtmux4_size*", -hier) + find(design, "xtmux4e_size*", -
hier) + find(design, "xtmux4p_size*", -hier) >/dev/null
XTBOOTH = find(design, "xtbooth*", -hier) >/dev/null
XTADD = find(design, "xtinc*", -hier) + find(design, "xtadd*", -hier) +
find(design, "xtcsa_size*", -hier) + find(design, "xtrelational*", -hier)
>/dev/null
XTMUL = find(design, "xtmul*", -hier) + find(design, "xtmac*", -hier)
>/dev/null
XTREGFILE = find(design, "xtregfile*", -hier) >/dev/null

/* set the compilation order */
XTPRIM = XTGATE + XTCLOCKGATE + XTRFLATCH + XTMUX2 + XTMUX3 + XTMUX4 + XTBOOTH
+ XTADD + XTMUL + XTREGFILE

/* set compile options */
XTFLATTEN = {}
XTSTRUCTURE = {}
XTDONT_TOUCH = XTCLOCKGATE + XTREGFILE
XTINCREMENTAL = XTADD + XTMUL + XTREGFILE
XTAREA = XTCLOCKGATE + XTRFLATCH
XTRELAXED = XTREGFILE


/*===========================================================================
    Premap the primitives
 ===========================================================================*/

if (XTFLATTEN != {}) {
    set_flatten true -design XTFLATTEN
}
if (XTPRIM - XTSTRUCTURE != {}) {
    set_structure false -design XTPRIM - XTSTRUCTURE
}
if (XTDONT_TOUCH != {}) {
    set_dont_touch XTDONT_TOUCH true
}

foreach (D, XTPRIM) {
    echo "Primitive map " + D
    current_design D

    echo "Ungrouping " + D
    ungroup -all -flatten >/dev/null

    echo "Constraining " + D
    if (({D} - XTAREA) == {}) {
      echo D + ": Area optimization"
      set_max_area 0
    } else {
      if (({D} - XTRELAXED) == {}) {
          /* normal constraints */
          CLOCK_PERIOD = XTCLOCK_PERIOD
          CLOCK_SKEW = XTCLOCK_SKEW
      } else {
          /* overconstrain all other primitives */
```

```
        CLOCK_PERIOD = 0
        CLOCK_SKEW = 0
    }
    echo D + ": Clock period is " + CLOCK_PERIOD + " and clock skew is " +
CLOCK_SKEW
/*
 +-----------------------------------------------------------------+
 |          Copyright (c) 1997-2000 Tensilica Inc.                  |
 |                                                                  |
 |   These coded instructions, statements, and computer programs   |
 |   are Confidential Proprietary Information of Tensilica Inc.    |
 |   and may not be disclosed to third parties or copied in any    |
 |   form, in whole or in part, without the prior written          |
 |   consent of Tensilica Inc.                                      |
 +-----------------------------------------------------------------+

  Title:    Generic Design Compiler Constraints

  Created:  November, 1998

;#  Author:       Richard Rudell
;#           <rudell@tensilica.com>

  Description:


  Revision History:
      Nov 1999: Changed multicycle paths for RFLATCH into a
        set_disable_timing on the latches instead

      Nov 1998: Original version

*/

/* ==================== Clocks ==================== */
CLOCK_PORT = find(port, "CLK") + find(port, "G*CLK") + find(port, "clk")
>/dev/null
if (CLOCK_PORT == {}) {
    create_clock -name CLK -period CLOCK_PERIOD
} else {
    CLOCK_PORT = filter(CLOCK_PORT, "@port_direction == in") >/dev/null
    create_clock CLOCK_PORT -name CLK -period CLOCK_PERIOD
}
set_dont_touch_network find(clock, "*")
set_fix_hold find(clock, "*")
set_clock_skew -ideal -uncertainty CLOCK_SKEW find(clock, "*")
DEBUG_CLOCK_PORT = find(port, "TClockDR") >/dev/null
if (DEBUG_CLOCK_PORT != {}) {
    create_clock DEBUG_CLOCK_PORT -name TClockDR -period 4 * CLOCK_PERIOD
}


/* ==================== I/O delays, loads, drives ==================== */
set_input_delay .20 * CLOCK_PERIOD -clock CLK all_inputs() - CLOCK_PORT -
DEBUG_CLOCK_PORT
set_output_delay .20 * CLOCK_PERIOD -clock CLK all_outputs()
set_load {4 * load_of(BOUNDARY_LOAD)} all_outputs()
```

```
set_driving_cell -cell DRIVE_CELL -pin DRIVE_PIN -from_pin DRIVE_PIN_FROM
all_inputs() - CLOCK_PORT - DEBUG_CLOCK_PORT >/dev/null


/* ==================== Miscellaneous ==================== */
set_operating_conditions OPERATING_CONDITION
/* BACKWARD COMPATIBILITY ISSUE: set_wire_load_model DOES NOT work with DC98.08
*/
/* set_wire_load_model -name WIRE_LOAD */
set_wire_load WIRE_LOAD
set_critical_range CRITICAL_RANGE current_design


/* ==================== Clock Gating Checks ==================== */
set_clock_gating_check -setup CLOCK_SKEW -hold CLOCK_SKEW current_design


/* ==================== Disable latch timing ==================== */
/* the if prevents RFLATCH from being printed */
if (FOOBAR == FOOBAR) {
    RFLATCH = find(cell, "*xtRF*latchout*", -hier) >/dev/null
    if (RFLATCH != {}) {
      echo disabling timing through the latches
      set_disable_timing RFLATCH
    }
}

/* ==================== False paths ==================== */
/*
if (DEBUG_CLOCK_PORT != {}) {
    set_false_path -from TClockDR -to CLK
    set_false_path -from CLK -to TClockDR
}
*/
      if (({D} - XTREGFILE) == {}) {
          set_input_delay .35 * CLOCK_PERIOD -clock CLK find(port, "wr*_addr")
>/dev/null
          set_input_delay .35 * CLOCK_PERIOD -clock CLK find(port, "wr*_we")
      }
    }

    echo "Optimizing " + D
    if (({D} - XTINCREMENTAL) == {}) {
      compile -map_effort low -ungroup_all -no_design_rule -incremental
    } else {
      compile -map_effort low -ungroup_all -no_design_rule
    }

    if (XTVERBOSE) {
      echo "Reporting " + D
      report_constraint
      report_timing
      report_area
      report_reference

      ELAPSE_TIME = time() - LAST_TIME
      LAST_TIME = time()
```

```
            echo D + " elapse time is " + ELAPSE_TIME
            echo D + " total time is " + time()
            echo D + " memory is " + mem()
        }
    }

    echo "Prim total time is " + time()
    echo "Prim memory is " + mem()

    remove_design find(design, "xtmux3e_1024") >/dev/null
    remove_design find(design, "xtmux4e_1024") >/dev/null
    remove_design find(design, "xtcsa_1024") >/dev/null

    current_design XTCURRENT_DESIGN
    CLOCK_PERIOD = XTCLOCK_PERIOD
    CLOCK_SKEW = XTCLOCK_SKEW
    /*
      +------------------------------------------------------------------+
      |            Copyright (c) 1997-2000 Tensilica Inc.                |
      |                                                                  |
      |    These coded instructions, statements, and computer programs   |
      |    are Confidential Proprietary Information of Tensilica Inc.    |
      |    and may not be disclosed to third parties or copied in any    |
      |    form, in whole or in part, without the prior written          |
      |    consent of Tensilica Inc.                                     |
      +------------------------------------------------------------------+

      Title:    Synthesis script for TIE Coprocessors

      Created:  Fri Nov 12, 1999

    ;#  Author:        Richard Rudell
    ;#             <rudell@tensilica.com>

      Description:

        Controls Design Compiler for optimizing TIE Coprocessors.

        Set TIE_DESIGN to TIE to optimize the TIE module, or set it to the verilog
        name of a semantic block (e.g., TIE_vec_mac) to optimize just that
        module.

        Set TIE_RETIME to 1 to perform retiming ("optimize_registers").  All of the
        TIE logic except for the the pipelined register files will be retimed.  If
        TIE_RETIME is 2, only the register file cores will not be retimed.  This
        allows for retiming of the pipeline logic within the register files, but is
        more taxing on the Design Compiler retiming algorithm.

        TIE_MAP_EFFORT is one of {low, medium, high} for the final optimization.

        The steps are as follows:
          - group the top-level logic into a design (TIE_toplogic)
          - set compile options
          - optimize the design for each top-level cell (low effort)
          - TIE_RETIME: regroup the top-level design for retiming
          - optimize top-level design (using TIE_MAP_EFFORT)
          - TIE_RETIME: retime the top-level design
```

```
    - optimize top-level design (using TIE_MAP_EFFORT)
    - fix design rules

  Revision History:
    Nov 1999: Original version

*/


/*=============================================================================
    Group the TIE top-level logic into a subdesign
 ============================================================================*/
current_design TIE_DESIGN
if (TIE_UNGROUP != {}) {
    /* remove some cells */
    ungroup TIE_UNGROUP -flatten
}
if (TIE_DESIGN == "xmTIE") {
    /* group the top-level random logic into a subdesign */
    TIE_CELL_LIST = find(cell, "TIE_*") >/dev/null
    group -design_name xmTIE_toplogic -cell_name TIE_toplogic -except
TIE_CELL_LIST
}


/*=============================================================================
    Find the top-level cells and their designs
 ============================================================================*/
current_design TIE_DESIGN
if (TIE_DESIGN == "xmTIE") {
    TIE_CELL_LIST = find(cell, "TIE_*") >/dev/null
    TIE_DESIGN_LIST = {}
} else {
    TIE_CELL_LIST = {}
    TIE_DESIGN_LIST = TIE_DESIGN
}
foreach (C, TIE_CELL_LIST) {
    TIE_DESIGN_LIST = TIE_DESIGN_LIST + find(design, "xm" + C)
}
TIE_REGFILE = find(design, "xmTIE*_Regfile", -hier) + find(design,
"xmTIE*_State", -hier) >/dev/null
TIE_XTREGFILE = find(design, "xtregfile*", -hier) >/dev/null
TIE_DECODER = find(design, "xmTIE_decoder", -hier) >/dev/null


/*=============================================================================
    Set optimization controls.
 ============================================================================*/
TIE_FLATTEN = TIE_DECODER           /* always flatten decoder */
if (AREA_IS_PRIORITY) {
    TIE_STRUCTURE = TIE_DESIGN_LIST
} else {
    TIE_STRUCTURE = TIE_DECODER           /* always structure decoder */
}
if (TIE_FLATTEN != {}) {
    set_flatten true -effort medium -design TIE_FLATTEN
}
```

```
if (TIE_DESIGN_LIST - TIE_STRUCTURE != {}) {
    set_structure false -design TIE_DESIGN_LIST - TIE_STRUCTURE
}


/*============================================================================
    Premap the hierarchical designs
 ============================================================================*/
LAST_TIME = time()
foreach (D, TIE_DESIGN_LIST) {
    echo "Premapping " + D
    current_design D

    echo "Ungrouping " + D
    ungroup -all -flatten

    echo "Constraining " + D
    set_resource_allocation none
    set_resource_implementation area_only
/*
 +------------------------------------------------------------------+
 |          Copyright (c) 1997-2000 Tensilica Inc.                  |
 |                                                                  |
 |   These coded instructions, statements, and computer programs    |
 |   are Confidential Proprietary Information of Tensilica Inc.     |
 |   and may not be disclosed to third parties or copied in any     |
 |   form, in whole or in part, without the prior written           |
 |   consent of Tensilica Inc.                                      |
 +------------------------------------------------------------------+

  Title:    Generic Design Compiler Constraints

  Created:  November, 1998

;#  Author:        Richard Rudell
;#           <rudell@tensilica.com>

  Description:


  Revision History:
      Nov 1999: Changed multicycle paths for RFLATCH into a
        set_disable_timing on the latches instead

      Nov 1998: Original version

*/

/* ==================== Clocks ==================== */
CLOCK_PORT = find(port, "CLK") + find(port, "G*CLK") + find(port, "clk")
>/dev/null
if (CLOCK_PORT == {}) {
    create_clock -name CLK -period CLOCK_PERIOD
} else {
    CLOCK_PORT = filter(CLOCK_PORT, "@port_direction == in") >/dev/null
    create_clock CLOCK_PORT -name CLK -period CLOCK_PERIOD
}
```

```
set_dont_touch_network find(clock, "*")
set_fix_hold find(clock, "*")
set_clock_skew -ideal -uncertainty CLOCK_SKEW find(clock, "*")
DEBUG_CLOCK_PORT = find(port, "TClockDR") >/dev/null
if (DEBUG_CLOCK_PORT != {}) {
    create_clock DEBUG_CLOCK_PORT -name TClockDR -period 4 * CLOCK_PERIOD
}


/* ==================== I/O delays, loads, drives ==================== */
set_input_delay .20 * CLOCK_PERIOD -clock CLK all_inputs() - CLOCK_PORT -
DEBUG_CLOCK_PORT
set_output_delay .20 * CLOCK_PERIOD -clock CLK all_outputs()
set_load {4 * load_of(BOUNDARY_LOAD)} all_outputs()
set_driving_cell -cell DRIVE_CELL -pin DRIVE_PIN -from_pin DRIVE_PIN_FROM
all_inputs() - CLOCK_PORT - DEBUG_CLOCK_PORT >/dev/null


/* ==================== Miscellaneous ==================== */
set_operating_conditions OPERATING_CONDITION
/* BACKWARD COMPATIBILITY ISSUE: set_wire_load_model DOES NOT work with DC98.08
*/
/* set_wire_load_model -name WIRE_LOAD */
set_wire_load WIRE_LOAD
set_critical_range CRITICAL_RANGE current_design


/* ==================== Clock Gating Checks ==================== */
set_clock_gating_check -setup CLOCK_SKEW -hold CLOCK_SKEW current_design


/* ==================== Disable latch timing ==================== */
/* the if prevents RFLATCH from being printed */
if (FOOBAR == FOOBAR) {
    RFLATCH = find(cell, "*xtRF*latchout*", -hier) >/dev/null
    if (RFLATCH != {}) {
      echo disabling timing through the latches
      set_disable_timing RFLATCH
    }
}

/* ==================== False paths ==================== */
/*
if (DEBUG_CLOCK_PORT != {}) {
    set_false_path -from TClockDR -to CLK
    set_false_path -from CLK -to TClockDR
}
*/
if (FOOBAR == FOOBAR) {
    X = find(port, "MemOpAddr_E") >/dev/null
    if (X != {}) {
      echo setting input delay for TIE memory interface
      set_input_delay .50 * CLOCK_PERIOD -clock CLK X
    }
    X = find(port, "TIE_MemLoadData_M") + find(port, "MemDataIn*") >/dev/null
    if (X != {}) {
      echo setting input delay for TIE memory interface
```

```
      set_input_delay .60 * CLOCK_PERIOD -clock CLK X
    }

    /* constraints for TIE register files and TIE state */
    X = find(port, "rd*_data_C*") + find(port, "ps_data_C*") >/dev/null
    if (X != {}) {
      echo setting output delay for TIE register file
      set_output_delay .95 * CLOCK_PERIOD -clock CLK X
    }
    X = find(port, "wd*_data*_C*") + find(port, "wr*_data*_C*") + find(port,
"ns_data*_C*") >/dev/null
    if (X != {}) {
      echo setting input delay for TIE register file
      set_input_delay .90 * CLOCK_PERIOD -clock CLK X
    }
    X = find(port, "wd*_wen_C*") + find(port, "Kill*") >/dev/null
    if (X != {}) {
      X = filter(X, "@port_direction == in") >/dev/null
      if (X != {}) {
          echo setting input delay for TIE register file controls
          set_input_delay .35 * CLOCK_PERIOD -clock CLK X
      }
    }
  }
    if (TIE_RETIME) {
      set_critical_range CLOCK_PERIOD current_design
    }

    echo "Optimizing " + D
    compile -map_effort low -ungroup_all -no_design_rule

    echo "Reporting " + D
    report_constraint
    report_timing
    report_area
    report_reference

    ELAPSE_TIME = time() - LAST_TIME
    LAST_TIME = time()
    echo D + " elapse time is " + ELAPSE_TIME
    echo D + " total time is " + time()
    echo D + " memory is " + mem()
}

echo "Premap total time is " + time()
echo "Premap memory is " + mem()


/*=========================================================================
    Report on the top level
 =========================================================================*/
current_design TIE_DESIGN
/*
 +---------------------------------------------------------------+
 |          Copyright (c) 1997-2000 Tensilica Inc.               |
 |                                                               |
 |   These coded instructions, statements, and computer programs |
```

```
  Title:    Generic Design Compiler Constraints

  Created:  November, 1998

;#  Author:       Richard Rudell
;#          <rudell@tensilica.com>

  Description:


  Revision History:
      Nov 1999: Changed multicycle paths for RFLATCH into a
        set_disable_timing on the latches instead

      Nov 1998: Original version

*/

/* ==================== Clocks ==================== */
CLOCK_PORT = find(port, "CLK") + find(port, "G*CLK") + find(port, "clk")
>/dev/null
if (CLOCK_PORT == {}) {
    create_clock -name CLK -period CLOCK_PERIOD
} else {
    CLOCK_PORT = filter(CLOCK_PORT, "@port_direction == in") >/dev/null
    create_clock CLOCK_PORT -name CLK -period CLOCK_PERIOD
}
set_dont_touch_network find(clock, "*")
set_fix_hold find(clock, "*")
set_clock_skew -ideal -uncertainty CLOCK_SKEW find(clock, "*")
DEBUG_CLOCK_PORT = find(port, "TClockDR") >/dev/null
if (DEBUG_CLOCK_PORT != {}) {
    create_clock DEBUG_CLOCK_PORT -name TClockDR -period 4 * CLOCK_PERIOD
}


/* ==================== I/O delays, loads, drives ==================== */
set_input_delay .20 * CLOCK_PERIOD -clock CLK all_inputs() - CLOCK_PORT -
DEBUG_CLOCK_PORT
set_output_delay .20 * CLOCK_PERIOD -clock CLK all_outputs()
set_load {4 * load_of(BOUNDARY_LOAD)} all_outputs()
set_driving_cell -cell DRIVE_CELL -pin DRIVE_PIN -from_pin DRIVE_PIN_FROM
all_inputs() - CLOCK_PORT - DEBUG_CLOCK_PORT >/dev/null


/* ==================== Miscellaneous ==================== */
set_operating_conditions OPERATING_CONDITION
/* BACKWARD COMPATIBILITY ISSUE: set_wire_load_model DOES NOT work with DC98.08
*/
/* set_wire_load_model -name WIRE_LOAD */
set_wire_load WIRE_LOAD
```

```
set_critical_range CRITICAL_RANGE current_design


/* =================== Clock Gating Checks =================== */
set_clock_gating_check -setup CLOCK_SKEW -hold CLOCK_SKEW current_design


/* =================== Disable latch timing =================== */
/* the if prevents RFLATCH from being printed */
if (FOOBAR == FOOBAR) {
    RFLATCH = find(cell, "*xtRF*latchout*", -hier) >/dev/null
    if (RFLATCH != {}) {
      echo disabling timing through the latches
      set_disable_timing RFLATCH
    }
}

/* =================== False paths =================== */
/*
if (DEBUG_CLOCK_PORT != {}) {
    set_false_path -from TClockDR -to CLK
    set_false_path -from CLK -to TClockDR
}
*/
if (FOOBAR == FOOBAR) {
    X = find(port, "MemOpAddr_E") >/dev/null
    if (X != {}) {
      echo setting input delay for TIE memory interface
      set_input_delay .50 * CLOCK_PERIOD -clock CLK X
    }
    X = find(port, "TIE_MemLoadData_M") + find(port, "MemDataIn*") >/dev/null
    if (X != {}) {
      echo setting input delay for TIE memory interface
      set_input_delay .60 * CLOCK_PERIOD -clock CLK X
    }

    /* constraints for TIE register files and TIE state */
    X = find(port, "rd*_data_C*") + find(port, "ps_data_C*") >/dev/null
    if (X != {}) {
      echo setting output delay for TIE register file
      set_output_delay .95 * CLOCK_PERIOD -clock CLK X
    }
    X = find(port, "wd*_data*_C*") + find(port, "wr*_data*_C*") + find(port,
"ns_data*_C*") >/dev/null
    if (X != {}) {
      echo setting input delay for TIE register file
      set_input_delay .90 * CLOCK_PERIOD -clock CLK X
    }
    X = find(port, "wd*_wen_C*") + find(port, "Kill*") >/dev/null
    if (X != {}) {
      X = filter(X, "@port_direction == in") >/dev/null
      if (X != {}) {
          echo setting input delay for TIE register file controls
          set_input_delay .35 * CLOCK_PERIOD -clock CLK X
      }
    }
}
```

```
report_constraint
report_timing
report_area
report_reference


/*=========================================================================
    Prepare design for retiming: keep the register files as subdesigns,
    and group everything else into "datapath".  Also, set a very high
    critical range so that all paths are made fast.
 =========================================================================*/
current_design TIE_DESIGN
if (TIE_RETIME) {
    set_critical_range CLOCK_PERIOD current_design

    if (TIE_RETIME == 2) {
      TIE_KEEP_DESIGN = TIE_XTREGFILE
    } else {
      TIE_KEEP_DESIGN = TIE_REGFILE
    }
    list TIE_KEEP_DESIGN
    if (TIE_KEEP_DESIGN == {}) {
      TIE_RETIME_DESIGN = TIE_DESIGN
      ungroup -all -flatten
    } else {
      TIE_RETIME_DESIGN = "xmTIE_datapath"
      set_dont_touch TIE_KEEP_DESIGN true
      ungroup -all -flatten
      set_dont_touch TIE_KEEP_DESIGN false
      if (TIE_RETIME == 2) {
          TIE_KEEP_CELL = find(cell, "*icore")
      } else {
          TIE_KEEP_CELL = find(cell, "TIE*_Regfile") + find(cell, "TIE*_State")
      }
      group -design TIE_RETIME_DESIGN -cell TIE_RETIME_DESIGN -except
TIE_KEEP_CELL
      list TIE_KEEP_CELL
    }
}


/*=========================================================================
    Pass 1
 =========================================================================*/
current_design TIE_DESIGN
if (TIE_XTREGFILE != {}) {
    set_dont_touch TIE_XTREGFILE false
}
if (TIE_DESIGN == "xmTIE") {
    compile_no_new_cells_at_top_level = true
}
uniquify
compile -incremental -map_effort TIE_MAP_EFFORT -no_design_rule -
boundary_optimization
report_constraint
report_timing
report_area
```

```
report_reference
ELAPSE_TIME = time() - LAST_TIME
LAST_TIME = time()
echo "pass1 elapse time is " + ELAPSE_TIME
echo "pass1 total time is " + time()
echo "pass1 memory is " + mem()


/*=============================================================================
    Retime
 ============================================================================*/
current_design TIE_DESIGN
if (TIE_RETIME) {
    if (TIE_RETIME_DESIGN != TIE_DESIGN) {
      characterize TIE_RETIME_DESIGN
      current_design TIE_RETIME_DESIGN
      set_wire_load WIRE_LOAD
    }
    optimize_registers -check_design -print_critical_loop -no_incremental_map
    current_design TIE_DESIGN
    set_critical_range CRITICAL_RANGE current_design
}


/*=============================================================================
    Pass 2 (add area constraint)
 ============================================================================*/
current_design TIE_DESIGN
set_max_area 0
compile -incremental -map_effort TIE_MAP_EFFORT -no_design_rule -
boundary_optimization
report_constraint
report_timing
report_area
report_reference
ELAPSE_TIME = time() - LAST_TIME
LAST_TIME = time()
echo "pass2 elapse time is " + ELAPSE_TIME
echo "pass2 total time is " + time()
echo "pass2 memory is " + mem()


/*=============================================================================
    Pass 3 (Design Rules)
 ============================================================================*/
current_design TIE_DESIGN
compile -incremental -map_effort TIE_MAP_EFFORT -only_design_rule -
boundary_optimization
report_constraint
report_timing
report_area
report_reference
ELAPSE_TIME = time() - LAST_TIME
LAST_TIME = time()
echo "pass3 elapse time is " + ELAPSE_TIME
echo "pass3 total time is " + time()
echo "pass3 memory is " + mem()
```

```
/*=========================================================================
    Write it out
 =========================================================================*/
current_design TIE_DESIGN
write -o TIE_DESIGN + ".db" -hier


/*=========================================================================
    Final hierarchical area/timing report
 =========================================================================*/
current_design TIE_DESIGN
X = find(cell, "TIE_*") + find(cell, "icore") >/dev/null
if (X != {}) {
    characterize X
}

current_design TIE_DESIGN
report_hierarchy > TIE_DESIGN + ".report"
foreach (D, TIE_DESIGN + find(design, "*", -hier)) {
    echo "Final report " + D
    current_design D
    report_constraint >> TIE_DESIGN + ".report"
    report_timing >> TIE_DESIGN + ".report"
    report_area >> TIE_DESIGN + ".report"
    report_reference >> TIE_DESIGN + ".report"
}
echo "xmTIE elapse time is " + time() >>TIE_DESIGN + ".report"
echo "xmTIE memory is " + mem() >>TIE_DESIGN + ".report"

sh rm -rf workdir
echo "xmTIE total time is " + time()
echo "xmTIE memory is " + mem()
quit
```

prim.v

```
// +------------------------------------------------------------------+
// |            Copyright (c) 1997-2000 Tensilica Inc.                  |
// |                                                                    |
// |   These coded instructions, statements, and computer programs     |
// |   are Confidential Proprietary Information of Tensilica Inc.      |
// |   and may not be disclosed to third parties or copied in any      |
// |   form, in whole or in part, without the prior written            |
// |   consent of Tensilica Inc.                                        |
// +------------------------------------------------------------------+
//
//  Title:  Base Synthesis Primitives
//
//  Created:      Tue Sep 28 16:59:24 1999
//
//
//  Description:
//
//  Revision History:
```

```
//

module xtmux3e(xtout, a, b, c, sel);
parameter size = 32;
output [size-1:0] xtout;
input [size-1:0] a, b, c;
input [1:0] sel;
    wire [1023:0] tmp;
    wire [1023:0] fa;
    wire [1023:0] fb;
    wire [1023:0] fc;
            assign fa[1023:size] = {(1024 - size){1'b0}};
            assign fa[size-1:0] = a;
            assign fb[1023:size] = {(1024 - size){1'b0}};
            assign fb[size-1:0] = b;
            assign fc[1023:size] = {(1024 - size){1'b0}};
            assign fc[size-1:0] = c;
    xtmux3e_1024 i(tmp, fa, fb, fc, sel);
    assign xtout = tmp;
endmodule

module xtmux3b(xtout, a, b, c, sel);
output xtout;
input a, b, c;
input [1:0] sel;
   // synopsys infer_mux "xtmux3b"
   assign xtout = sel[1] ? c : (sel[0] ? b : a);
endmodule

module xtmux4e(xtout, a, b, c, d, sel);
parameter size = 32;
output [size-1:0] xtout;
input [size-1:0] a, b, c, d;
input [1:0] sel;
    wire [1023:0] tmp;
    wire [1023:0] fa;
    wire [1023:0] fb;
    wire [1023:0] fc;
    wire [1023:0] fd;
            assign fa[1023:size] = {(1024 - size){1'b0}};
            assign fa[size-1:0] = a;
            assign fb[1023:size] = {(1024 - size){1'b0}};
            assign fb[size-1:0] = b;
            assign fc[1023:size] = {(1024 - size){1'b0}};
            assign fc[size-1:0] = c;
            assign fd[1023:size] = {(1024 - size){1'b0}};
            assign fd[size-1:0] = d;
    xtmux4e_1024 i(tmp, fa, fb, fc, fd, sel);
    assign xtout = tmp;
endmodule

module xtmux4b(xtout, a, b, c, d, sel);
output xtout;
input a, b, c, d;
input [1:0] sel;
    GTECH_MUX4 i(.D0(a), .D1(b), .D2(c), .D3(d), .A(sel[0]), .B(sel[1]),
.Z(xtout));
```

```
endmodule

module xtcsa(sum, carry, a, b, c);
parameter size = 32;
output [size-1:0] sum, carry;
input [size-1:0] a, b, c;
    wire [1023:0] tmp1, tmp2;
    wire [1023:0] fa;
    wire [1023:0] fb;
    wire [1023:0] fc;
            assign fa[1023:size] = {(1024 - size){1'b0}};
            assign fa[size-1:0] = a;
            assign fb[1023:size] = {(1024 - size){1'b0}};
            assign fb[size-1:0] = b;
            assign fc[1023:size] = {(1024 - size){1'b0}};
            assign fc[size-1:0] = c;
    xtcsa_1024 i(tmp1, tmp2, fa, fb, fc);
    assign sum = tmp1;
    assign carry = tmp2;
endmodule


module xtfa(sum, carry, a, b, c);
output sum, carry;
input a, b, c;
    GTECH_ADD_ABC i(a, b, c, sum, carry);
endmodule


module xtha(sum, carry, a, b);
output sum, carry;
input a, b;
    GTECH_ADD_AB i(a, b, sum, carry);
endmodule

module xtmux3e_1024(xtout, a, b, c, sel);
output [1023:0] xtout;
input [1023:0] a, b, c;
input [1:0] sel;
    xtmux3b i0(.xtout(xtout[0]), .a(a[0]), .b(b[0]), .c(c[0]), .sel(sel));
    xtmux3b i1(.xtout(xtout[1]), .a(a[1]), .b(b[1]), .c(c[1]), .sel(sel));
    xtmux3b i2(.xtout(xtout[2]), .a(a[2]), .b(b[2]), .c(c[2]), .sel(sel));
    xtmux3b i3(.xtout(xtout[3]), .a(a[3]), .b(b[3]), .c(c[3]), .sel(sel));
    xtmux3b i4(.xtout(xtout[4]), .a(a[4]), .b(b[4]), .c(c[4]), .sel(sel));
    xtmux3b i5(.xtout(xtout[5]), .a(a[5]), .b(b[5]), .c(c[5]), .sel(sel));
    xtmux3b i6(.xtout(xtout[6]), .a(a[6]), .b(b[6]), .c(c[6]), .sel(sel));
    xtmux3b i7(.xtout(xtout[7]), .a(a[7]), .b(b[7]), .c(c[7]), .sel(sel));
    xtmux3b i8(.xtout(xtout[8]), .a(a[8]), .b(b[8]), .c(c[8]), .sel(sel));
    xtmux3b i9(.xtout(xtout[9]), .a(a[9]), .b(b[9]), .c(c[9]), .sel(sel));
    xtmux3b i10(.xtout(xtout[10]), .a(a[10]), .b(b[10]), .c(c[10]), .sel(sel));
    xtmux3b i11(.xtout(xtout[11]), .a(a[11]), .b(b[11]), .c(c[11]), .sel(sel));
    xtmux3b i12(.xtout(xtout[12]), .a(a[12]), .b(b[12]), .c(c[12]), .sel(sel));
    xtmux3b i13(.xtout(xtout[13]), .a(a[13]), .b(b[13]), .c(c[13]), .sel(sel));
    xtmux3b i14(.xtout(xtout[14]), .a(a[14]), .b(b[14]), .c(c[14]), .sel(sel));
    xtmux3b i15(.xtout(xtout[15]), .a(a[15]), .b(b[15]), .c(c[15]), .sel(sel));
    xtmux3b i16(.xtout(xtout[16]), .a(a[16]), .b(b[16]), .c(c[16]), .sel(sel));
    xtmux3b i17(.xtout(xtout[17]), .a(a[17]), .b(b[17]), .c(c[17]), .sel(sel));
```

```
xtmux3b i18(.xtout(xtout[18]), .a(a[18]), .b(b[18]), .c(c[18]), .sel(sel));
xtmux3b i19(.xtout(xtout[19]), .a(a[19]), .b(b[19]), .c(c[19]), .sel(sel));
xtmux3b i20(.xtout(xtout[20]), .a(a[20]), .b(b[20]), .c(c[20]), .sel(sel));
xtmux3b i21(.xtout(xtout[21]), .a(a[21]), .b(b[21]), .c(c[21]), .sel(sel));
xtmux3b i22(.xtout(xtout[22]), .a(a[22]), .b(b[22]), .c(c[22]), .sel(sel));
xtmux3b i23(.xtout(xtout[23]), .a(a[23]), .b(b[23]), .c(c[23]), .sel(sel));
xtmux3b i24(.xtout(xtout[24]), .a(a[24]), .b(b[24]), .c(c[24]), .sel(sel));
xtmux3b i25(.xtout(xtout[25]), .a(a[25]), .b(b[25]), .c(c[25]), .sel(sel));
xtmux3b i26(.xtout(xtout[26]), .a(a[26]), .b(b[26]), .c(c[26]), .sel(sel));
xtmux3b i27(.xtout(xtout[27]), .a(a[27]), .b(b[27]), .c(c[27]), .sel(sel));
xtmux3b i28(.xtout(xtout[28]), .a(a[28]), .b(b[28]), .c(c[28]), .sel(sel));
xtmux3b i29(.xtout(xtout[29]), .a(a[29]), .b(b[29]), .c(c[29]), .sel(sel));
xtmux3b i30(.xtout(xtout[30]), .a(a[30]), .b(b[30]), .c(c[30]), .sel(sel));
xtmux3b i31(.xtout(xtout[31]), .a(a[31]), .b(b[31]), .c(c[31]), .sel(sel));
xtmux3b i32(.xtout(xtout[32]), .a(a[32]), .b(b[32]), .c(c[32]), .sel(sel));
xtmux3b i33(.xtout(xtout[33]), .a(a[33]), .b(b[33]), .c(c[33]), .sel(sel));
xtmux3b i34(.xtout(xtout[34]), .a(a[34]), .b(b[34]), .c(c[34]), .sel(sel));
xtmux3b i35(.xtout(xtout[35]), .a(a[35]), .b(b[35]), .c(c[35]), .sel(sel));
xtmux3b i36(.xtout(xtout[36]), .a(a[36]), .b(b[36]), .c(c[36]), .sel(sel));
xtmux3b i37(.xtout(xtout[37]), .a(a[37]), .b(b[37]), .c(c[37]), .sel(sel));
xtmux3b i38(.xtout(xtout[38]), .a(a[38]), .b(b[38]), .c(c[38]), .sel(sel));
xtmux3b i39(.xtout(xtout[39]), .a(a[39]), .b(b[39]), .c(c[39]), .sel(sel));
xtmux3b i40(.xtout(xtout[40]), .a(a[40]), .b(b[40]), .c(c[40]), .sel(sel));
xtmux3b i41(.xtout(xtout[41]), .a(a[41]), .b(b[41]), .c(c[41]), .sel(sel));
xtmux3b i42(.xtout(xtout[42]), .a(a[42]), .b(b[42]), .c(c[42]), .sel(sel));
xtmux3b i43(.xtout(xtout[43]), .a(a[43]), .b(b[43]), .c(c[43]), .sel(sel));
xtmux3b i44(.xtout(xtout[44]), .a(a[44]), .b(b[44]), .c(c[44]), .sel(sel));
xtmux3b i45(.xtout(xtout[45]), .a(a[45]), .b(b[45]), .c(c[45]), .sel(sel));
xtmux3b i46(.xtout(xtout[46]), .a(a[46]), .b(b[46]), .c(c[46]), .sel(sel));
xtmux3b i47(.xtout(xtout[47]), .a(a[47]), .b(b[47]), .c(c[47]), .sel(sel));
xtmux3b i48(.xtout(xtout[48]), .a(a[48]), .b(b[48]), .c(c[48]), .sel(sel));
xtmux3b i49(.xtout(xtout[49]), .a(a[49]), .b(b[49]), .c(c[49]), .sel(sel));
xtmux3b i50(.xtout(xtout[50]), .a(a[50]), .b(b[50]), .c(c[50]), .sel(sel));
xtmux3b i51(.xtout(xtout[51]), .a(a[51]), .b(b[51]), .c(c[51]), .sel(sel));
xtmux3b i52(.xtout(xtout[52]), .a(a[52]), .b(b[52]), .c(c[52]), .sel(sel));
xtmux3b i53(.xtout(xtout[53]), .a(a[53]), .b(b[53]), .c(c[53]), .sel(sel));
xtmux3b i54(.xtout(xtout[54]), .a(a[54]), .b(b[54]), .c(c[54]), .sel(sel));
xtmux3b i55(.xtout(xtout[55]), .a(a[55]), .b(b[55]), .c(c[55]), .sel(sel));
xtmux3b i56(.xtout(xtout[56]), .a(a[56]), .b(b[56]), .c(c[56]), .sel(sel));
xtmux3b i57(.xtout(xtout[57]), .a(a[57]), .b(b[57]), .c(c[57]), .sel(sel));
xtmux3b i58(.xtout(xtout[58]), .a(a[58]), .b(b[58]), .c(c[58]), .sel(sel));
xtmux3b i59(.xtout(xtout[59]), .a(a[59]), .b(b[59]), .c(c[59]), .sel(sel));
xtmux3b i60(.xtout(xtout[60]), .a(a[60]), .b(b[60]), .c(c[60]), .sel(sel));
xtmux3b i61(.xtout(xtout[61]), .a(a[61]), .b(b[61]), .c(c[61]), .sel(sel));
xtmux3b i62(.xtout(xtout[62]), .a(a[62]), .b(b[62]), .c(c[62]), .sel(sel));
xtmux3b i63(.xtout(xtout[63]), .a(a[63]), .b(b[63]), .c(c[63]), .sel(sel));
xtmux3b i64(.xtout(xtout[64]), .a(a[64]), .b(b[64]), .c(c[64]), .sel(sel));
xtmux3b i65(.xtout(xtout[65]), .a(a[65]), .b(b[65]), .c(c[65]), .sel(sel));
xtmux3b i66(.xtout(xtout[66]), .a(a[66]), .b(b[66]), .c(c[66]), .sel(sel));
xtmux3b i67(.xtout(xtout[67]), .a(a[67]), .b(b[67]), .c(c[67]), .sel(sel));
xtmux3b i68(.xtout(xtout[68]), .a(a[68]), .b(b[68]), .c(c[68]), .sel(sel));
xtmux3b i69(.xtout(xtout[69]), .a(a[69]), .b(b[69]), .c(c[69]), .sel(sel));
xtmux3b i70(.xtout(xtout[70]), .a(a[70]), .b(b[70]), .c(c[70]), .sel(sel));
xtmux3b i71(.xtout(xtout[71]), .a(a[71]), .b(b[71]), .c(c[71]), .sel(sel));
xtmux3b i72(.xtout(xtout[72]), .a(a[72]), .b(b[72]), .c(c[72]), .sel(sel));
xtmux3b i73(.xtout(xtout[73]), .a(a[73]), .b(b[73]), .c(c[73]), .sel(sel));
xtmux3b i74(.xtout(xtout[74]), .a(a[74]), .b(b[74]), .c(c[74]), .sel(sel));
```

```
        xtmux3b i75(.xtout(xtout[75]), .a(a[75]), .b(b[75]), .c(c[75]), .sel(sel));
        xtmux3b i76(.xtout(xtout[76]), .a(a[76]), .b(b[76]), .c(c[76]), .sel(sel));
        xtmux3b i77(.xtout(xtout[77]), .a(a[77]), .b(b[77]), .c(c[77]), .sel(sel));
        xtmux3b i78(.xtout(xtout[78]), .a(a[78]), .b(b[78]), .c(c[78]), .sel(sel));
        xtmux3b i79(.xtout(xtout[79]), .a(a[79]), .b(b[79]), .c(c[79]), .sel(sel));
        xtmux3b i80(.xtout(xtout[80]), .a(a[80]), .b(b[80]), .c(c[80]), .sel(sel));
        xtmux3b i81(.xtout(xtout[81]), .a(a[81]), .b(b[81]), .c(c[81]), .sel(sel));
        xtmux3b i82(.xtout(xtout[82]), .a(a[82]), .b(b[82]), .c(c[82]), .sel(sel));
        xtmux3b i83(.xtout(xtout[83]), .a(a[83]), .b(b[83]), .c(c[83]), .sel(sel));
        xtmux3b i84(.xtout(xtout[84]), .a(a[84]), .b(b[84]), .c(c[84]), .sel(sel));
        xtmux3b i85(.xtout(xtout[85]), .a(a[85]), .b(b[85]), .c(c[85]), .sel(sel));
        xtmux3b i86(.xtout(xtout[86]), .a(a[86]), .b(b[86]), .c(c[86]), .sel(sel));
        xtmux3b i87(.xtout(xtout[87]), .a(a[87]), .b(b[87]), .c(c[87]), .sel(sel));
        xtmux3b i88(.xtout(xtout[88]), .a(a[88]), .b(b[88]), .c(c[88]), .sel(sel));
        xtmux3b i89(.xtout(xtout[89]), .a(a[89]), .b(b[89]), .c(c[89]), .sel(sel));
        xtmux3b i90(.xtout(xtout[90]), .a(a[90]), .b(b[90]), .c(c[90]), .sel(sel));
        xtmux3b i91(.xtout(xtout[91]), .a(a[91]), .b(b[91]), .c(c[91]), .sel(sel));
        xtmux3b i92(.xtout(xtout[92]), .a(a[92]), .b(b[92]), .c(c[92]), .sel(sel));
        xtmux3b i93(.xtout(xtout[93]), .a(a[93]), .b(b[93]), .c(c[93]), .sel(sel));
        xtmux3b i94(.xtout(xtout[94]), .a(a[94]), .b(b[94]), .c(c[94]), .sel(sel));
        xtmux3b i95(.xtout(xtout[95]), .a(a[95]), .b(b[95]), .c(c[95]), .sel(sel));
        xtmux3b i96(.xtout(xtout[96]), .a(a[96]), .b(b[96]), .c(c[96]), .sel(sel));
        xtmux3b i97(.xtout(xtout[97]), .a(a[97]), .b(b[97]), .c(c[97]), .sel(sel));
        xtmux3b i98(.xtout(xtout[98]), .a(a[98]), .b(b[98]), .c(c[98]), .sel(sel));
        xtmux3b i99(.xtout(xtout[99]), .a(a[99]), .b(b[99]), .c(c[99]), .sel(sel));
        xtmux3b i100(.xtout(xtout[100]), .a(a[100]), .b(b[100]), .c(c[100]),
.sel(sel));
        xtmux3b i101(.xtout(xtout[101]), .a(a[101]), .b(b[101]), .c(c[101]),
.sel(sel));
        xtmux3b i102(.xtout(xtout[102]), .a(a[102]), .b(b[102]), .c(c[102]),
.sel(sel));
        xtmux3b i103(.xtout(xtout[103]), .a(a[103]), .b(b[103]), .c(c[103]),
.sel(sel));
        xtmux3b i104(.xtout(xtout[104]), .a(a[104]), .b(b[104]), .c(c[104]),
.sel(sel));
        xtmux3b i105(.xtout(xtout[105]), .a(a[105]), .b(b[105]), .c(c[105]),
.sel(sel));
        xtmux3b i106(.xtout(xtout[106]), .a(a[106]), .b(b[106]), .c(c[106]),
.sel(sel));
        xtmux3b i107(.xtout(xtout[107]), .a(a[107]), .b(b[107]), .c(c[107]),
.sel(sel));
        xtmux3b i108(.xtout(xtout[108]), .a(a[108]), .b(b[108]), .c(c[108]),
.sel(sel));
        xtmux3b i109(.xtout(xtout[109]), .a(a[109]), .b(b[109]), .c(c[109]),
.sel(sel));
        xtmux3b i110(.xtout(xtout[110]), .a(a[110]), .b(b[110]), .c(c[110]),
.sel(sel));
        xtmux3b i111(.xtout(xtout[111]), .a(a[111]), .b(b[111]), .c(c[111]),
.sel(sel));
        xtmux3b i112(.xtout(xtout[112]), .a(a[112]), .b(b[112]), .c(c[112]),
.sel(sel));
        xtmux3b i113(.xtout(xtout[113]), .a(a[113]), .b(b[113]), .c(c[113]),
.sel(sel));
        xtmux3b i114(.xtout(xtout[114]), .a(a[114]), .b(b[114]), .c(c[114]),
.sel(sel));
        xtmux3b i115(.xtout(xtout[115]), .a(a[115]), .b(b[115]), .c(c[115]),
.sel(sel));
```

```
    xtmux3b i116(.xtout(xtout[116]), .a(a[116]), .b(b[116]), .c(c[116]),
.sel(sel));
    xtmux3b i117(.xtout(xtout[117]), .a(a[117]), .b(b[117]), .c(c[117]),
.sel(sel));
    xtmux3b i118(.xtout(xtout[118]), .a(a[118]), .b(b[118]), .c(c[118]),
.sel(sel));
    xtmux3b i119(.xtout(xtout[119]), .a(a[119]), .b(b[119]), .c(c[119]),
.sel(sel));
    xtmux3b i120(.xtout(xtout[120]), .a(a[120]), .b(b[120]), .c(c[120]),
.sel(sel));
    xtmux3b i121(.xtout(xtout[121]), .a(a[121]), .b(b[121]), .c(c[121]),
.sel(sel));
    xtmux3b i122(.xtout(xtout[122]), .a(a[122]), .b(b[122]), .c(c[122]),
.sel(sel));
    xtmux3b i123(.xtout(xtout[123]), .a(a[123]), .b(b[123]), .c(c[123]),
.sel(sel));
    xtmux3b i124(.xtout(xtout[124]), .a(a[124]), .b(b[124]), .c(c[124]),
.sel(sel));
    xtmux3b i125(.xtout(xtout[125]), .a(a[125]), .b(b[125]), .c(c[125]),
.sel(sel));
    xtmux3b i126(.xtout(xtout[126]), .a(a[126]), .b(b[126]), .c(c[126]),
.sel(sel));
    xtmux3b i127(.xtout(xtout[127]), .a(a[127]), .b(b[127]), .c(c[127]),
.sel(sel));
    xtmux3b i128(.xtout(xtout[128]), .a(a[128]), .b(b[128]), .c(c[128]),
.sel(sel));
    xtmux3b i129(.xtout(xtout[129]), .a(a[129]), .b(b[129]), .c(c[129]),
.sel(sel));
    xtmux3b i130(.xtout(xtout[130]), .a(a[130]), .b(b[130]), .c(c[130]),
.sel(sel));
    xtmux3b i131(.xtout(xtout[131]), .a(a[131]), .b(b[131]), .c(c[131]),
.sel(sel));
    xtmux3b i132(.xtout(xtout[132]), .a(a[132]), .b(b[132]), .c(c[132]),
.sel(sel));
    xtmux3b i133(.xtout(xtout[133]), .a(a[133]), .b(b[133]), .c(c[133]),
.sel(sel));
    xtmux3b i134(.xtout(xtout[134]), .a(a[134]), .b(b[134]), .c(c[134]),
.sel(sel));
    xtmux3b i135(.xtout(xtout[135]), .a(a[135]), .b(b[135]), .c(c[135]),
.sel(sel));
    xtmux3b i136(.xtout(xtout[136]), .a(a[136]), .b(b[136]), .c(c[136]),
.sel(sel));
    xtmux3b i137(.xtout(xtout[137]), .a(a[137]), .b(b[137]), .c(c[137]),
.sel(sel));
    xtmux3b i138(.xtout(xtout[138]), .a(a[138]), .b(b[138]), .c(c[138]),
.sel(sel));
    xtmux3b i139(.xtout(xtout[139]), .a(a[139]), .b(b[139]), .c(c[139]),
.sel(sel));
    xtmux3b i140(.xtout(xtout[140]), .a(a[140]), .b(b[140]), .c(c[140]),
.sel(sel));
    xtmux3b i141(.xtout(xtout[141]), .a(a[141]), .b(b[141]), .c(c[141]),
.sel(sel));
    xtmux3b i142(.xtout(xtout[142]), .a(a[142]), .b(b[142]), .c(c[142]),
.sel(sel));
    xtmux3b i143(.xtout(xtout[143]), .a(a[143]), .b(b[143]), .c(c[143]),
.sel(sel));
```

```
xtmux3b i144(.xtout(xtout[144]), .a(a[144]), .b(b[144]), .c(c[144]),
.sel(sel));
xtmux3b i145(.xtout(xtout[145]), .a(a[145]), .b(b[145]), .c(c[145]),
.sel(sel));
xtmux3b i146(.xtout(xtout[146]), .a(a[146]), .b(b[146]), .c(c[146]),
.sel(sel));
xtmux3b i147(.xtout(xtout[147]), .a(a[147]), .b(b[147]), .c(c[147]),
.sel(sel));
xtmux3b i148(.xtout(xtout[148]), .a(a[148]), .b(b[148]), .c(c[148]),
.sel(sel));
xtmux3b i149(.xtout(xtout[149]), .a(a[149]), .b(b[149]), .c(c[149]),
.sel(sel));
xtmux3b i150(.xtout(xtout[150]), .a(a[150]), .b(b[150]), .c(c[150]),
.sel(sel));
xtmux3b i151(.xtout(xtout[151]), .a(a[151]), .b(b[151]), .c(c[151]),
.sel(sel));
xtmux3b i152(.xtout(xtout[152]), .a(a[152]), .b(b[152]), .c(c[152]),
.sel(sel));
xtmux3b i153(.xtout(xtout[153]), .a(a[153]), .b(b[153]), .c(c[153]),
.sel(sel));
xtmux3b i154(.xtout(xtout[154]), .a(a[154]), .b(b[154]), .c(c[154]),
.sel(sel));
xtmux3b i155(.xtout(xtout[155]), .a(a[155]), .b(b[155]), .c(c[155]),
.sel(sel));
xtmux3b i156(.xtout(xtout[156]), .a(a[156]), .b(b[156]), .c(c[156]),
.sel(sel));
xtmux3b i157(.xtout(xtout[157]), .a(a[157]), .b(b[157]), .c(c[157]),
.sel(sel));
xtmux3b i158(.xtout(xtout[158]), .a(a[158]), .b(b[158]), .c(c[158]),
.sel(sel));
xtmux3b i159(.xtout(xtout[159]), .a(a[159]), .b(b[159]), .c(c[159]),
.sel(sel));
xtmux3b i160(.xtout(xtout[160]), .a(a[160]), .b(b[160]), .c(c[160]),
.sel(sel));
xtmux3b i161(.xtout(xtout[161]), .a(a[161]), .b(b[161]), .c(c[161]),
.sel(sel));
xtmux3b i162(.xtout(xtout[162]), .a(a[162]), .b(b[162]), .c(c[162]),
.sel(sel));
xtmux3b i163(.xtout(xtout[163]), .a(a[163]), .b(b[163]), .c(c[163]),
.sel(sel));
xtmux3b i164(.xtout(xtout[164]), .a(a[164]), .b(b[164]), .c(c[164]),
.sel(sel));
xtmux3b i165(.xtout(xtout[165]), .a(a[165]), .b(b[165]), .c(c[165]),
.sel(sel));
xtmux3b i166(.xtout(xtout[166]), .a(a[166]), .b(b[166]), .c(c[166]),
.sel(sel));
xtmux3b i167(.xtout(xtout[167]), .a(a[167]), .b(b[167]), .c(c[167]),
.sel(sel));
xtmux3b i168(.xtout(xtout[168]), .a(a[168]), .b(b[168]), .c(c[168]),
.sel(sel));
xtmux3b i169(.xtout(xtout[169]), .a(a[169]), .b(b[169]), .c(c[169]),
.sel(sel));
xtmux3b i170(.xtout(xtout[170]), .a(a[170]), .b(b[170]), .c(c[170]),
.sel(sel));
xtmux3b i171(.xtout(xtout[171]), .a(a[171]), .b(b[171]), .c(c[171]),
.sel(sel));
```

```
xtmux3b i172(.xtout(xtout[172]), .a(a[172]), .b(b[172]), .c(c[172]),
.sel(sel));
xtmux3b i173(.xtout(xtout[173]), .a(a[173]), .b(b[173]), .c(c[173]),
.sel(sel));
xtmux3b i174(.xtout(xtout[174]), .a(a[174]), .b(b[174]), .c(c[174]),
.sel(sel));
xtmux3b i175(.xtout(xtout[175]), .a(a[175]), .b(b[175]), .c(c[175]),
.sel(sel));
xtmux3b i176(.xtout(xtout[176]), .a(a[176]), .b(b[176]), .c(c[176]),
.sel(sel));
xtmux3b i177(.xtout(xtout[177]), .a(a[177]), .b(b[177]), .c(c[177]),
.sel(sel));
xtmux3b i178(.xtout(xtout[178]), .a(a[178]), .b(b[178]), .c(c[178]),
.sel(sel));
xtmux3b i179(.xtout(xtout[179]), .a(a[179]), .b(b[179]), .c(c[179]),
.sel(sel));
xtmux3b i180(.xtout(xtout[180]), .a(a[180]), .b(b[180]), .c(c[180]),
.sel(sel));
xtmux3b i181(.xtout(xtout[181]), .a(a[181]), .b(b[181]), .c(c[181]),
.sel(sel));
xtmux3b i182(.xtout(xtout[182]), .a(a[182]), .b(b[182]), .c(c[182]),
.sel(sel));
xtmux3b i183(.xtout(xtout[183]), .a(a[183]), .b(b[183]), .c(c[183]),
.sel(sel));
xtmux3b i184(.xtout(xtout[184]), .a(a[184]), .b(b[184]), .c(c[184]),
.sel(sel));
xtmux3b i185(.xtout(xtout[185]), .a(a[185]), .b(b[185]), .c(c[185]),
.sel(sel));
xtmux3b i186(.xtout(xtout[186]), .a(a[186]), .b(b[186]), .c(c[186]),
.sel(sel));
xtmux3b i187(.xtout(xtout[187]), .a(a[187]), .b(b[187]), .c(c[187]),
.sel(sel));
xtmux3b i188(.xtout(xtout[188]), .a(a[188]), .b(b[188]), .c(c[188]),
.sel(sel));
xtmux3b i189(.xtout(xtout[189]), .a(a[189]), .b(b[189]), .c(c[189]),
.sel(sel));
xtmux3b i190(.xtout(xtout[190]), .a(a[190]), .b(b[190]), .c(c[190]),
.sel(sel));
xtmux3b i191(.xtout(xtout[191]), .a(a[191]), .b(b[191]), .c(c[191]),
.sel(sel));
xtmux3b i192(.xtout(xtout[192]), .a(a[192]), .b(b[192]), .c(c[192]),
.sel(sel));
xtmux3b i193(.xtout(xtout[193]), .a(a[193]), .b(b[193]), .c(c[193]),
.sel(sel));
xtmux3b i194(.xtout(xtout[194]), .a(a[194]), .b(b[194]), .c(c[194]),
.sel(sel));
xtmux3b i195(.xtout(xtout[195]), .a(a[195]), .b(b[195]), .c(c[195]),
.sel(sel));
xtmux3b i196(.xtout(xtout[196]), .a(a[196]), .b(b[196]), .c(c[196]),
.sel(sel));
xtmux3b i197(.xtout(xtout[197]), .a(a[197]), .b(b[197]), .c(c[197]),
.sel(sel));
xtmux3b i198(.xtout(xtout[198]), .a(a[198]), .b(b[198]), .c(c[198]),
.sel(sel));
xtmux3b i199(.xtout(xtout[199]), .a(a[199]), .b(b[199]), .c(c[199]),
.sel(sel));
```

```
    xtmux3b i200(.xtout(xtout[200]), .a(a[200]), .b(b[200]), .c(c[200]),
.sel(sel));
    xtmux3b i201(.xtout(xtout[201]), .a(a[201]), .b(b[201]), .c(c[201]),
.sel(sel));
    xtmux3b i202(.xtout(xtout[202]), .a(a[202]), .b(b[202]), .c(c[202]),
.sel(sel));
    xtmux3b i203(.xtout(xtout[203]), .a(a[203]), .b(b[203]), .c(c[203]),
.sel(sel));
    xtmux3b i204(.xtout(xtout[204]), .a(a[204]), .b(b[204]), .c(c[204]),
.sel(sel));
    xtmux3b i205(.xtout(xtout[205]), .a(a[205]), .b(b[205]), .c(c[205]),
.sel(sel));
    xtmux3b i206(.xtout(xtout[206]), .a(a[206]), .b(b[206]), .c(c[206]),
.sel(sel));
    xtmux3b i207(.xtout(xtout[207]), .a(a[207]), .b(b[207]), .c(c[207]),
.sel(sel));
    xtmux3b i208(.xtout(xtout[208]), .a(a[208]), .b(b[208]), .c(c[208]),
.sel(sel));
    xtmux3b i209(.xtout(xtout[209]), .a(a[209]), .b(b[209]), .c(c[209]),
.sel(sel));
    xtmux3b i210(.xtout(xtout[210]), .a(a[210]), .b(b[210]), .c(c[210]),
.sel(sel));
    xtmux3b i211(.xtout(xtout[211]), .a(a[211]), .b(b[211]), .c(c[211]),
.sel(sel));
    xtmux3b i212(.xtout(xtout[212]), .a(a[212]), .b(b[212]), .c(c[212]),
.sel(sel));
    xtmux3b i213(.xtout(xtout[213]), .a(a[213]), .b(b[213]), .c(c[213]),
.sel(sel));
    xtmux3b i214(.xtout(xtout[214]), .a(a[214]), .b(b[214]), .c(c[214]),
.sel(sel));
    xtmux3b i215(.xtout(xtout[215]), .a(a[215]), .b(b[215]), .c(c[215]),
.sel(sel));
    xtmux3b i216(.xtout(xtout[216]), .a(a[216]), .b(b[216]), .c(c[216]),
.sel(sel));
    xtmux3b i217(.xtout(xtout[217]), .a(a[217]), .b(b[217]), .c(c[217]),
.sel(sel));
    xtmux3b i218(.xtout(xtout[218]), .a(a[218]), .b(b[218]), .c(c[218]),
.sel(sel));
    xtmux3b i219(.xtout(xtout[219]), .a(a[219]), .b(b[219]), .c(c[219]),
.sel(sel));
    xtmux3b i220(.xtout(xtout[220]), .a(a[220]), .b(b[220]), .c(c[220]),
.sel(sel));
    xtmux3b i221(.xtout(xtout[221]), .a(a[221]), .b(b[221]), .c(c[221]),
.sel(sel));
    xtmux3b i222(.xtout(xtout[222]), .a(a[222]), .b(b[222]), .c(c[222]),
.sel(sel));
    xtmux3b i223(.xtout(xtout[223]), .a(a[223]), .b(b[223]), .c(c[223]),
.sel(sel));
    xtmux3b i224(.xtout(xtout[224]), .a(a[224]), .b(b[224]), .c(c[224]),
.sel(sel));
    xtmux3b i225(.xtout(xtout[225]), .a(a[225]), .b(b[225]), .c(c[225]),
.sel(sel));
    xtmux3b i226(.xtout(xtout[226]), .a(a[226]), .b(b[226]), .c(c[226]),
.sel(sel));
    xtmux3b i227(.xtout(xtout[227]), .a(a[227]), .b(b[227]), .c(c[227]),
.sel(sel));
```

```
    xtmux3b i228(.xtout(xtout[228]), .a(a[228]), .b(b[228]), .c(c[228]),
.sel(sel));
    xtmux3b i229(.xtout(xtout[229]), .a(a[229]), .b(b[229]), .c(c[229]),
.sel(sel));
    xtmux3b i230(.xtout(xtout[230]), .a(a[230]), .b(b[230]), .c(c[230]),
.sel(sel));
    xtmux3b i231(.xtout(xtout[231]), .a(a[231]), .b(b[231]), .c(c[231]),
.sel(sel));
    xtmux3b i232(.xtout(xtout[232]), .a(a[232]), .b(b[232]), .c(c[232]),
.sel(sel));
    xtmux3b i233(.xtout(xtout[233]), .a(a[233]), .b(b[233]), .c(c[233]),
.sel(sel));
    xtmux3b i234(.xtout(xtout[234]), .a(a[234]), .b(b[234]), .c(c[234]),
.sel(sel));
    xtmux3b i235(.xtout(xtout[235]), .a(a[235]), .b(b[235]), .c(c[235]),
.sel(sel));
    xtmux3b i236(.xtout(xtout[236]), .a(a[236]), .b(b[236]), .c(c[236]),
.sel(sel));
    xtmux3b i237(.xtout(xtout[237]), .a(a[237]), .b(b[237]), .c(c[237]),
.sel(sel));
    xtmux3b i238(.xtout(xtout[238]), .a(a[238]), .b(b[238]), .c(c[238]),
..sel(sel));
    xtmux3b i239(.xtout(xtout[239]), .a(a[239]), .b(b[239]), .c(c[239]),
.sel(sel));
    xtmux3b i240(.xtout(xtout[240]), .a(a[240]), .b(b[240]), .c(c[240]),
.sel(sel));
    xtmux3b i241(.xtout(xtout[241]), .a(a[241]), .b(b[241]), .c(c[241]),
.sel(sel));
    xtmux3b i242(.xtout(xtout[242]), .a(a[242]), .b(b[242]), .c(c[242]),
.sel(sel));
    xtmux3b i243(.xtout(xtout[243]), .a(a[243]), .b(b[243]), .c(c[243]),
.sel(sel));
    xtmux3b i244(.xtout(xtout[244]), .a(a[244]), .b(b[244]), .c(c[244]),
.sel(sel));
    xtmux3b i245(.xtout(xtout[245]), .a(a[245]), .b(b[245]), .c(c[245]),
.sel(sel));
    xtmux3b i246(.xtout(xtout[246]), .a(a[246]), .b(b[246]), .c(c[246]),
.sel(sel));
    xtmux3b i247(.xtout(xtout[247]), .a(a[247]), .b(b[247]), .c(c[247]),
.sel(sel));
    xtmux3b i248(.xtout(xtout[248]), .a(a[248]), .b(b[248]), .c(c[248]),
.sel(sel));
    xtmux3b i249(.xtout(xtout[249]), .a(a[249]), .b(b[249]), .c(c[249]),
.sel(sel));
    xtmux3b i250(.xtout(xtout[250]), .a(a[250]), .b(b[250]), .c(c[250]),
.sel(sel));
    xtmux3b i251(.xtout(xtout[251]), .a(a[251]), .b(b[251]), .c(c[251]),
.sel(sel));
    xtmux3b i252(.xtout(xtout[252]), .a(a[252]), .b(b[252]), .c(c[252]),
.sel(sel));
    xtmux3b i253(.xtout(xtout[253]), .a(a[253]), .b(b[253]), .c(c[253]),
.sel(sel));
    xtmux3b i254(.xtout(xtout[254]), .a(a[254]), .b(b[254]), .c(c[254]),
.sel(sel));
    xtmux3b i255(.xtout(xtout[255]), .a(a[255]), .b(b[255]), .c(c[255]),
.sel(sel));
```

```
    xtmux3b i256(.xtout(xtout[256]), .a(a[256]), .b(b[256]), .c(c[256]),
.sel(sel));
    xtmux3b i257(.xtout(xtout[257]), .a(a[257]), .b(b[257]), .c(c[257]),
.sel(sel));
    xtmux3b i258(.xtout(xtout[258]), .a(a[258]), .b(b[258]), .c(c[258]),
.sel(sel));
    xtmux3b i259(.xtout(xtout[259]), .a(a[259]), .b(b[259]), .c(c[259]),
.sel(sel));
    xtmux3b i260(.xtout(xtout[260]), .a(a[260]), .b(b[260]), .c(c[260]),
.sel(sel));
    xtmux3b i261(.xtout(xtout[261]), .a(a[261]), .b(b[261]), .c(c[261]),
.sel(sel));
    xtmux3b i262(.xtout(xtout[262]), .a(a[262]), .b(b[262]), .c(c[262]),
.sel(sel));
    xtmux3b i263(.xtout(xtout[263]), .a(a[263]), .b(b[263]), .c(c[263]),
.sel(sel));
    xtmux3b i264(.xtout(xtout[264]), .a(a[264]), .b(b[264]), .c(c[264]),
.sel(sel));
    xtmux3b i265(.xtout(xtout[265]), .a(a[265]), .b(b[265]), .c(c[265]),
.sel(sel));
    xtmux3b i266(.xtout(xtout[266]), .a(a[266]), .b(b[266]), .c(c[266]),
.sel(sel));
    xtmux3b i267(.xtout(xtout[267]), .a(a[267]), .b(b[267]), .c(c[267]),
.sel(sel));
    xtmux3b i268(.xtout(xtout[268]), .a(a[268]), .b(b[268]), .c(c[268]),
.sel(sel));
    xtmux3b i269(.xtout(xtout[269]), .a(a[269]), .b(b[269]), .c(c[269]),
.sel(sel));
    xtmux3b i270(.xtout(xtout[270]), .a(a[270]), .b(b[270]), .c(c[270]),
.sel(sel));
    xtmux3b i271(.xtout(xtout[271]), .a(a[271]), .b(b[271]), .c(c[271]),
.sel(sel));
    xtmux3b i272(.xtout(xtout[272]), .a(a[272]), .b(b[272]), .c(c[272]),
.sel(sel));
    xtmux3b i273(.xtout(xtout[273]), .a(a[273]), .b(b[273]), .c(c[273]),
.sel(sel));
    xtmux3b i274(.xtout(xtout[274]), .a(a[274]), .b(b[274]), .c(c[274]),
.sel(sel));
    xtmux3b i275(.xtout(xtout[275]), .a(a[275]), .b(b[275]), .c(c[275]),
.sel(sel));
    xtmux3b i276(.xtout(xtout[276]), .a(a[276]), .b(b[276]), .c(c[276]),
.sel(sel));
    xtmux3b i277(.xtout(xtout[277]), .a(a[277]), .b(b[277]), .c(c[277]),
.sel(sel));
    xtmux3b i278(.xtout(xtout[278]), .a(a[278]), .b(b[278]), .c(c[278]),
.sel(sel));
    xtmux3b i279(.xtout(xtout[279]), .a(a[279]), .b(b[279]), .c(c[279]),
.sel(sel));
    xtmux3b i280(.xtout(xtout[280]), .a(a[280]), .b(b[280]), .c(c[280]),
.sel(sel));
    xtmux3b i281(.xtout(xtout[281]), .a(a[281]), .b(b[281]), .c(c[281]),
.sel(sel));
    xtmux3b i282(.xtout(xtout[282]), .a(a[282]), .b(b[282]), .c(c[282]),
.sel(sel));
    xtmux3b i283(.xtout(xtout[283]), .a(a[283]), .b(b[283]), .c(c[283]),
.sel(sel));
```

```
    xtmux3b i284(.xtout(xtout[284]), .a(a[284]), .b(b[284]), .c(c[284]),
.sel(sel));
    xtmux3b i285(.xtout(xtout[285]), .a(a[285]), .b(b[285]), .c(c[285]),
.sel(sel));
    xtmux3b i286(.xtout(xtout[286]), .a(a[286]), .b(b[286]), .c(c[286]),
.sel(sel));
    xtmux3b i287(.xtout(xtout[287]), .a(a[287]), .b(b[287]), .c(c[287]),
.sel(sel));
    xtmux3b i288(.xtout(xtout[288]), .a(a[288]), .b(b[288]), .c(c[288]),
.sel(sel));
    xtmux3b i289(.xtout(xtout[289]), .a(a[289]), .b(b[289]), .c(c[289]),
.sel(sel));
    xtmux3b i290(.xtout(xtout[290]), .a(a[290]), .b(b[290]), .c(c[290]),
.sel(sel));
    xtmux3b i291(.xtout(xtout[291]), .a(a[291]), .b(b[291]), .c(c[291]),
.sel(sel));
    xtmux3b i292(.xtout(xtout[292]), .a(a[292]), .b(b[292]), .c(c[292]),
.sel(sel));
    xtmux3b i293(.xtout(xtout[293]), .a(a[293]), .b(b[293]), .c(c[293]),
.sel(sel));
    xtmux3b i294(.xtout(xtout[294]), .a(a[294]), .b(b[294]), .c(c[294]),
.sel(sel));
    xtmux3b i295(.xtout(xtout[295]), .a(a[295]), .b(b[295]), .c(c[295]),
.sel(sel));
    xtmux3b i296(.xtout(xtout[296]), .a(a[296]), .b(b[296]), .c(c[296]),
.sel(sel));
    xtmux3b i297(.xtout(xtout[297]), .a(a[297]), .b(b[297]), .c(c[297]),
.sel(sel));
    xtmux3b i298(.xtout(xtout[298]), .a(a[298]), .b(b[298]), .c(c[298]),
.sel(sel));
    xtmux3b i299(.xtout(xtout[299]), .a(a[299]), .b(b[299]), .c(c[299]),
.sel(sel));
    xtmux3b i300(.xtout(xtout[300]), .a(a[300]), .b(b[300]), .c(c[300]),
.sel(sel));
    xtmux3b i301(.xtout(xtout[301]), .a(a[301]), .b(b[301]), .c(c[301]),
.sel(sel));
    xtmux3b i302(.xtout(xtout[302]), .a(a[302]), .b(b[302]), .c(c[302]),
.sel(sel));
    xtmux3b i303(.xtout(xtout[303]), .a(a[303]), .b(b[303]), .c(c[303]),
.sel(sel));
    xtmux3b i304(.xtout(xtout[304]), .a(a[304]), .b(b[304]), .c(c[304]),
.sel(sel));
    xtmux3b i305(.xtout(xtout[305]), .a(a[305]), .b(b[305]), .c(c[305]),
.sel(sel));
    xtmux3b i306(.xtout(xtout[306]), .a(a[306]), .b(b[306]), .c(c[306]),
.sel(sel));
    xtmux3b i307(.xtout(xtout[307]), .a(a[307]), .b(b[307]), .c(c[307]),
.sel(sel));
    xtmux3b i308(.xtout(xtout[308]), .a(a[308]), .b(b[308]), .c(c[308]),
.sel(sel));
    xtmux3b i309(.xtout(xtout[309]), .a(a[309]), .b(b[309]), .c(c[309]),
.sel(sel));
    xtmux3b i310(.xtout(xtout[310]), .a(a[310]), .b(b[310]), .c(c[310]),
.sel(sel));
    xtmux3b i311(.xtout(xtout[311]), .a(a[311]), .b(b[311]), .c(c[311]),
.sel(sel));
```

```
    xtmux3b i312(.xtout(xtout[312]), .a(a[312]), .b(b[312]), .c(c[312]),
.sel(sel));
    xtmux3b i313(.xtout(xtout[313]), .a(a[313]), .b(b[313]), .c(c[313]),
.sel(sel));
    xtmux3b i314(.xtout(xtout[314]), .a(a[314]), .b(b[314]), .c(c[314]),
.sel(sel));
    xtmux3b i315(.xtout(xtout[315]), .a(a[315]), .b(b[315]), .c(c[315]),
.sel(sel));
    xtmux3b i316(.xtout(xtout[316]), .a(a[316]), .b(b[316]), .c(c[316]),
.sel(sel));
    xtmux3b i317(.xtout(xtout[317]), .a(a[317]), .b(b[317]), .c(c[317]),
.sel(sel));
    xtmux3b i318(.xtout(xtout[318]), .a(a[318]), .b(b[318]), .c(c[318]),
.sel(sel));
    xtmux3b i319(.xtout(xtout[319]), .a(a[319]), .b(b[319]), .c(c[319]),
.sel(sel));
    xtmux3b i320(.xtout(xtout[320]), .a(a[320]), .b(b[320]), .c(c[320]),
.sel(sel));
    xtmux3b i321(.xtout(xtout[321]), .a(a[321]), .b(b[321]), .c(c[321]),
.sel(sel));
    xtmux3b i322(.xtout(xtout[322]), .a(a[322]), .b(b[322]), .c(c[322]),
.sel(sel));
    xtmux3b i323(.xtout(xtout[323]), .a(a[323]), .b(b[323]), .c(c[323]),
.sel(sel));
    xtmux3b i324(.xtout(xtout[324]), .a(a[324]), .b(b[324]), .c(c[324]),
.sel(sel));
    xtmux3b i325(.xtout(xtout[325]), .a(a[325]), .b(b[325]), .c(c[325]),
.sel(sel));
    xtmux3b i326(.xtout(xtout[326]), .a(a[326]), .b(b[326]), .c(c[326]),
.sel(sel));
    xtmux3b i327(.xtout(xtout[327]), .a(a[327]), .b(b[327]), .c(c[327]),
.sel(sel));
    xtmux3b i328(.xtout(xtout[328]), .a(a[328]), .b(b[328]), .c(c[328]),
.sel(sel));
    xtmux3b i329(.xtout(xtout[329]), .a(a[329]), .b(b[329]), .c(c[329]),
.sel(sel));
    xtmux3b i330(.xtout(xtout[330]), .a(a[330]), .b(b[330]), .c(c[330]),
.sel(sel));
    xtmux3b i331(.xtout(xtout[331]), .a(a[331]), .b(b[331]), .c(c[331]),
.sel(sel));
    xtmux3b i332(.xtout(xtout[332]), .a(a[332]), .b(b[332]), .c(c[332]),
.sel(sel));
    xtmux3b i333(.xtout(xtout[333]), .a(a[333]), .b(b[333]), .c(c[333]),
.sel(sel));
    xtmux3b i334(.xtout(xtout[334]), .a(a[334]), .b(b[334]), .c(c[334]),
.sel(sel));
    xtmux3b i335(.xtout(xtout[335]), .a(a[335]), .b(b[335]), .c(c[335]),
.sel(sel));
    xtmux3b i336(.xtout(xtout[336]), .a(a[336]), .b(b[336]), .c(c[336]),
.sel(sel));
    xtmux3b i337(.xtout(xtout[337]), .a(a[337]), .b(b[337]), .c(c[337]),
.sel(sel));
    xtmux3b i338(.xtout(xtout[338]), .a(a[338]), .b(b[338]), .c(c[338]),
.sel(sel));
    xtmux3b i339(.xtout(xtout[339]), .a(a[339]), .b(b[339]), .c(c[339]),
.sel(sel));
```

```
    xtmux3b i340(.xtout(xtout[340]), .a(a[340]), .b(b[340]), .c(c[340]),
.sel(sel));
    xtmux3b i341(.xtout(xtout[341]), .a(a[341]), .b(b[341]), .c(c[341]),
.sel(sel));
    xtmux3b i342(.xtout(xtout[342]), .a(a[342]), .b(b[342]), .c(c[342]),
.sel(sel));
    xtmux3b i343(.xtout(xtout[343]), .a(a[343]), .b(b[343]), .c(c[343]),
.sel(sel));
    xtmux3b i344(.xtout(xtout[344]), .a(a[344]), .b(b[344]), .c(c[344]),
.sel(sel));
    xtmux3b i345(.xtout(xtout[345]), .a(a[345]), .b(b[345]), .c(c[345]),
.sel(sel));
    xtmux3b i346(.xtout(xtout[346]), .a(a[346]), .b(b[346]), .c(c[346]),
.sel(sel));
    xtmux3b i347(.xtout(xtout[347]), .a(a[347]), .b(b[347]), .c(c[347]),
.sel(sel));
    xtmux3b i348(.xtout(xtout[348]), .a(a[348]), .b(b[348]), .c(c[348]),
.sel(sel));
    xtmux3b i349(.xtout(xtout[349]), .a(a[349]), .b(b[349]), .c(c[349]),
.sel(sel));
    xtmux3b i350(.xtout(xtout[350]), .a(a[350]), .b(b[350]), .c(c[350]),
.sel(sel));
    xtmux3b i351(.xtout(xtout[351]), .a(a[351]), .b(b[351]), .c(c[351]),
.sel(sel));
    xtmux3b i352(.xtout(xtout[352]), .a(a[352]), .b(b[352]), .c(c[352]),
.sel(sel));
    xtmux3b i353(.xtout(xtout[353]), .a(a[353]), .b(b[353]), .c(c[353]),
.sel(sel));
    xtmux3b i354(.xtout(xtout[354]), .a(a[354]), .b(b[354]), .c(c[354]),
.sel(sel));
    xtmux3b i355(.xtout(xtout[355]), .a(a[355]), .b(b[355]), .c(c[355]),
.sel(sel));
    xtmux3b i356(.xtout(xtout[356]), .a(a[356]), .b(b[356]), .c(c[356]),
.sel(sel));
    xtmux3b i357(.xtout(xtout[357]), .a(a[357]), .b(b[357]), .c(c[357]),
.sel(sel));
    xtmux3b i358(.xtout(xtout[358]), .a(a[358]), .b(b[358]), .c(c[358]),
.sel(sel));
    xtmux3b i359(.xtout(xtout[359]), .a(a[359]), .b(b[359]), .c(c[359]),
.sel(sel));
    xtmux3b i360(.xtout(xtout[360]), .a(a[360]), .b(b[360]), .c(c[360]),
.sel(sel));
    xtmux3b i361(.xtout(xtout[361]), .a(a[361]), .b(b[361]), .c(c[361]),
.sel(sel));
    xtmux3b i362(.xtout(xtout[362]), .a(a[362]), .b(b[362]), .c(c[362]),
.sel(sel));
    xtmux3b i363(.xtout(xtout[363]), .a(a[363]), .b(b[363]), .c(c[363]),
.sel(sel));
    xtmux3b i364(.xtout(xtout[364]), .a(a[364]), .b(b[364]), .c(c[364]),
.sel(sel));
    xtmux3b i365(.xtout(xtout[365]), .a(a[365]), .b(b[365]), .c(c[365]),
.sel(sel));
    xtmux3b i366(.xtout(xtout[366]), .a(a[366]), .b(b[366]), .c(c[366]),
.sel(sel));
    xtmux3b i367(.xtout(xtout[367]), .a(a[367]), .b(b[367]), .c(c[367]),
.sel(sel));
```

```
    xtmux3b i368(.xtout(xtout[368]), .a(a[368]), .b(b[368]), .c(c[368]),
.sel(sel));
    xtmux3b i369(.xtout(xtout[369]), .a(a[369]), .b(b[369]), .c(c[369]),
.sel(sel));
    xtmux3b i370(.xtout(xtout[370]), .a(a[370]), .b(b[370]), .c(c[370]),
.sel(sel));
    xtmux3b i371(.xtout(xtout[371]), .a(a[371]), .b(b[371]), .c(c[371]),
.sel(sel));
    xtmux3b i372(.xtout(xtout[372]), .a(a[372]), .b(b[372]), .c(c[372]),
.sel(sel));
    xtmux3b i373(.xtout(xtout[373]), .a(a[373]), .b(b[373]), .c(c[373]),
.sel(sel));
    xtmux3b i374(.xtout(xtout[374]), .a(a[374]), .b(b[374]), .c(c[374]),
.sel(sel));
    xtmux3b i375(.xtout(xtout[375]), .a(a[375]), .b(b[375]), .c(c[375]),
.sel(sel));
    xtmux3b i376(.xtout(xtout[376]), .a(a[376]), .b(b[376]), .c(c[376]),
.sel(sel));
    xtmux3b i377(.xtout(xtout[377]), .a(a[377]), .b(b[377]), .c(c[377]),
.sel(sel));
    xtmux3b i378(.xtout(xtout[378]), .a(a[378]), .b(b[378]), .c(c[378]),
.sel(sel));
    xtmux3b i379(.xtout(xtout[379]), .a(a[379]), .b(b[379]), .c(c[379]),
.sel(sel));
    xtmux3b i380(.xtout(xtout[380]), .a(a[380]), .b(b[380]), .c(c[380]),
.sel(sel));
    xtmux3b i381(.xtout(xtout[381]), .a(a[381]), .b(b[381]), .c(c[381]),
.sel(sel));
    xtmux3b i382(.xtout(xtout[382]), .a(a[382]), .b(b[382]), .c(c[382]),
.sel(sel));
    xtmux3b i383(.xtout(xtout[383]), .a(a[383]), .b(b[383]), .c(c[383]),
.sel(sel));
    xtmux3b i384(.xtout(xtout[384]), .a(a[384]), .b(b[384]), .c(c[384]),
.sel(sel));
    xtmux3b i385(.xtout(xtout[385]), .a(a[385]), .b(b[385]), .c(c[385]),
.sel(sel));
    xtmux3b i386(.xtout(xtout[386]), .a(a[386]), .b(b[386]), .c(c[386]),
.sel(sel));
    xtmux3b i387(.xtout(xtout[387]), .a(a[387]), .b(b[387]), .c(c[387]),
.sel(sel));
    xtmux3b i388(.xtout(xtout[388]), .a(a[388]), .b(b[388]), .c(c[388]),
.sel(sel));
    xtmux3b i389(.xtout(xtout[389]), .a(a[389]), .b(b[389]), .c(c[389]),
.sel(sel));
    xtmux3b i390(.xtout(xtout[390]), .a(a[390]), .b(b[390]), .c(c[390]),
.sel(sel));
    xtmux3b i391(.xtout(xtout[391]), .a(a[391]), .b(b[391]), .c(c[391]),
.sel(sel));
    xtmux3b i392(.xtout(xtout[392]), .a(a[392]), .b(b[392]), .c(c[392]),
.sel(sel));
    xtmux3b i393(.xtout(xtout[393]), .a(a[393]), .b(b[393]), .c(c[393]),
.sel(sel));
    xtmux3b i394(.xtout(xtout[394]), .a(a[394]), .b(b[394]), .c(c[394]),
.sel(sel));
    xtmux3b i395(.xtout(xtout[395]), .a(a[395]), .b(b[395]), .c(c[395]),
.sel(sel));
```

```
    xtmux3b i396(.xtout(xtout[396]), .a(a[396]), .b(b[396]), .c(c[396]),
.sel(sel));
    xtmux3b i397(.xtout(xtout[397]), .a(a[397]), .b(b[397]), .c(c[397]),
.sel(sel));
    xtmux3b i398(.xtout(xtout[398]), .a(a[398]), .b(b[398]), .c(c[398]),
.sel(sel));
    xtmux3b i399(.xtout(xtout[399]), .a(a[399]), .b(b[399]), .c(c[399]),
.sel(sel));
    xtmux3b i400(.xtout(xtout[400]), .a(a[400]), .b(b[400]), .c(c[400]),
.sel(sel));
    xtmux3b i401(.xtout(xtout[401]), .a(a[401]), .b(b[401]), .c(c[401]),
.sel(sel));
    xtmux3b i402(.xtout(xtout[402]), .a(a[402]), .b(b[402]), .c(c[402]),
.sel(sel));
    xtmux3b i403(.xtout(xtout[403]), .a(a[403]), .b(b[403]), .c(c[403]),
.sel(sel));
    xtmux3b i404(.xtout(xtout[404]), .a(a[404]), .b(b[404]), .c(c[404]),
.sel(sel));
    xtmux3b i405(.xtout(xtout[405]), .a(a[405]), .b(b[405]), .c(c[405]),
.sel(sel));
    xtmux3b i406(.xtout(xtout[406]), .a(a[406]), .b(b[406]), .c(c[406]),
.sel(sel));
    xtmux3b i407(.xtout(xtout[407]), .a(a[407]), .b(b[407]), .c(c[407]),
.sel(sel));
    xtmux3b i408(.xtout(xtout[408]), .a(a[408]), .b(b[408]), .c(c[408]),
.sel(sel));
    xtmux3b i409(.xtout(xtout[409]), .a(a[409]), .b(b[409]), .c(c[409]),
.sel(sel));
    xtmux3b i410(.xtout(xtout[410]), .a(a[410]), .b(b[410]), .c(c[410]),
.sel(sel));
    xtmux3b i411(.xtout(xtout[411]), .a(a[411]), .b(b[411]), .c(c[411]),
.sel(sel));
    xtmux3b i412(.xtout(xtout[412]), .a(a[412]), .b(b[412]), .c(c[412]),
.sel(sel));
    xtmux3b i413(.xtout(xtout[413]), .a(a[413]), .b(b[413]), .c(c[413]),
.sel(sel));
    xtmux3b i414(.xtout(xtout[414]), .a(a[414]), .b(b[414]), .c(c[414]),
.sel(sel));
    xtmux3b i415(.xtout(xtout[415]), .a(a[415]), .b(b[415]), .c(c[415]),
.sel(sel));
    xtmux3b i416(.xtout(xtout[416]), .a(a[416]), .b(b[416]), .c(c[416]),
.sel(sel));
    xtmux3b i417(.xtout(xtout[417]), .a(a[417]), .b(b[417]), .c(c[417]),
.sel(sel));
    xtmux3b i418(.xtout(xtout[418]), .a(a[418]), .b(b[418]), .c(c[418]),
.sel(sel));
    xtmux3b i419(.xtout(xtout[419]), .a(a[419]), .b(b[419]), .c(c[419]),
.sel(sel));
    xtmux3b i420(.xtout(xtout[420]), .a(a[420]), .b(b[420]), .c(c[420]),
.sel(sel));
    xtmux3b i421(.xtout(xtout[421]), .a(a[421]), .b(b[421]), .c(c[421]),
.sel(sel));
    xtmux3b i422(.xtout(xtout[422]), .a(a[422]), .b(b[422]), .c(c[422]),
.sel(sel));
    xtmux3b i423(.xtout(xtout[423]), .a(a[423]), .b(b[423]), .c(c[423]),
.sel(sel));
```

```
    xtmux3b i424(.xtout(xtout[424]), .a(a[424]), .b(b[424]), .c(c[424]),
.sel(sel));
    xtmux3b i425(.xtout(xtout[425]), .a(a[425]), .b(b[425]), .c(c[425]),
.sel(sel));
    xtmux3b i426(.xtout(xtout[426]), .a(a[426]), .b(b[426]), .c(c[426]),
.sel(sel));
    xtmux3b i427(.xtout(xtout[427]), .a(a[427]), .b(b[427]), .c(c[427]),
.sel(sel));
    xtmux3b i428(.xtout(xtout[428]), .a(a[428]), .b(b[428]), .c(c[428]),
.sel(sel));
    xtmux3b i429(.xtout(xtout[429]), .a(a[429]), .b(b[429]), .c(c[429]),
.sel(sel));
    xtmux3b i430(.xtout(xtout[430]), .a(a[430]), .b(b[430]), .c(c[430]),
.sel(sel));
    xtmux3b i431(.xtout(xtout[431]), .a(a[431]), .b(b[431]), .c(c[431]),
.sel(sel));
    xtmux3b i432(.xtout(xtout[432]), .a(a[432]), .b(b[432]), .c(c[432]),
.sel(sel));
    xtmux3b i433(.xtout(xtout[433]), .a(a[433]), .b(b[433]), .c(c[433]),
.sel(sel));
    xtmux3b i434(.xtout(xtout[434]), .a(a[434]), .b(b[434]), .c(c[434]),
.sel(sel));
    xtmux3b i435(.xtout(xtout[435]), .a(a[435]), .b(b[435]), .c(c[435]),
.sel(sel));
    xtmux3b i436(.xtout(xtout[436]), .a(a[436]), .b(b[436]), .c(c[436]),
.sel(sel));
    xtmux3b i437(.xtout(xtout[437]), .a(a[437]), .b(b[437]), .c(c[437]),
.sel(sel));
    xtmux3b i438(.xtout(xtout[438]), .a(a[438]), .b(b[438]), .c(c[438]),
.sel(sel));
    xtmux3b i439(.xtout(xtout[439]), .a(a[439]), .b(b[439]), .c(c[439]),
.sel(sel));
    xtmux3b i440(.xtout(xtout[440]), .a(a[440]), .b(b[440]), .c(c[440]),
.sel(sel));
    xtmux3b i441(.xtout(xtout[441]), .a(a[441]), .b(b[441]), .c(c[441]),
.sel(sel));
    xtmux3b i442(.xtout(xtout[442]), .a(a[442]), .b(b[442]), .c(c[442]),
.sel(sel));
    xtmux3b i443(.xtout(xtout[443]), .a(a[443]), .b(b[443]), .c(c[443]),
.sel(sel));
    xtmux3b i444(.xtout(xtout[444]), .a(a[444]), .b(b[444]), .c(c[444]),
.sel(sel));
    xtmux3b i445(.xtout(xtout[445]), .a(a[445]), .b(b[445]), .c(c[445]),
.sel(sel));
    xtmux3b i446(.xtout(xtout[446]), .a(a[446]), .b(b[446]), .c(c[446]),
.sel(sel));
    xtmux3b i447(.xtout(xtout[447]), .a(a[447]), .b(b[447]), .c(c[447]),
.sel(sel));
    xtmux3b i448(.xtout(xtout[448]), .a(a[448]), .b(b[448]), .c(c[448]),
.sel(sel));
    xtmux3b i449(.xtout(xtout[449]), .a(a[449]), .b(b[449]), .c(c[449]),
.sel(sel));
    xtmux3b i450(.xtout(xtout[450]), .a(a[450]), .b(b[450]), .c(c[450]),
.sel(sel));
    xtmux3b i451(.xtout(xtout[451]), .a(a[451]), .b(b[451]), .c(c[451]),
.sel(sel));
```

```
    xtmux3b i452(.xtout(xtout[452]), .a(a[452]), .b(b[452]), .c(c[452]),
.sel(sel));
    xtmux3b i453(.xtout(xtout[453]), .a(a[453]), .b(b[453]), .c(c[453]),
.sel(sel));
    xtmux3b i454(.xtout(xtout[454]), .a(a[454]), .b(b[454]), .c(c[454]),
.sel(sel));
    xtmux3b i455(.xtout(xtout[455]), .a(a[455]), .b(b[455]), .c(c[455]),
.sel(sel));
    xtmux3b i456(.xtout(xtout[456]), .a(a[456]), .b(b[456]), .c(c[456]),
.sel(sel));
    xtmux3b i457(.xtout(xtout[457]), .a(a[457]), .b(b[457]), .c(c[457]),
.sel(sel));
    xtmux3b i458(.xtout(xtout[458]), .a(a[458]), .b(b[458]), .c(c[458]),
.sel(sel));
    xtmux3b i459(.xtout(xtout[459]), .a(a[459]), .b(b[459]), .c(c[459]),
.sel(sel));
    xtmux3b i460(.xtout(xtout[460]), .a(a[460]), .b(b[460]), .c(c[460]),
.sel(sel));
    xtmux3b i461(.xtout(xtout[461]), .a(a[461]), .b(b[461]), .c(c[461]),
.sel(sel));
    xtmux3b i462(.xtout(xtout[462]), .a(a[462]), .b(b[462]), .c(c[462]),
.sel(sel));
    xtmux3b i463(.xtout(xtout[463]), .a(a[463]), .b(b[463]), .c(c[463]),
.sel(sel));
    xtmux3b i464(.xtout(xtout[464]), .a(a[464]), .b(b[464]), .c(c[464]),
.sel(sel));
    xtmux3b i465(.xtout(xtout[465]), .a(a[465]), .b(b[465]), .c(c[465]),
.sel(sel));
    xtmux3b i466(.xtout(xtout[466]), .a(a[466]), .b(b[466]), .c(c[466]),
.sel(sel));
    xtmux3b i467(.xtout(xtout[467]), .a(a[467]), .b(b[467]), .c(c[467]),
.sel(sel));
    xtmux3b i468(.xtout(xtout[468]), .a(a[468]), .b(b[468]), .c(c[468]),
.sel(sel));
    xtmux3b i469(.xtout(xtout[469]), .a(a[469]), .b(b[469]), .c(c[469]),
.sel(sel));
    xtmux3b i470(.xtout(xtout[470]), .a(a[470]), .b(b[470]), .c(c[470]),
.sel(sel));
    xtmux3b i471(.xtout(xtout[471]), .a(a[471]), .b(b[471]), .c(c[471]),
.sel(sel));
    xtmux3b i472(.xtout(xtout[472]), .a(a[472]), .b(b[472]), .c(c[472]),
.sel(sel));
    xtmux3b i473(.xtout(xtout[473]), .a(a[473]), .b(b[473]), .c(c[473]),
.sel(sel));
    xtmux3b i474(.xtout(xtout[474]), .a(a[474]), .b(b[474]), .c(c[474]),
.sel(sel));
    xtmux3b i475(.xtout(xtout[475]), .a(a[475]), .b(b[475]), .c(c[475]),
.sel(sel));
    xtmux3b i476(.xtout(xtout[476]), .a(a[476]), .b(b[476]), .c(c[476]),
.sel(sel));
    xtmux3b i477(.xtout(xtout[477]), .a(a[477]), .b(b[477]), .c(c[477]),
.sel(sel));
    xtmux3b i478(.xtout(xtout[478]), .a(a[478]), .b(b[478]), .c(c[478]),
.sel(sel));
    xtmux3b i479(.xtout(xtout[479]), .a(a[479]), .b(b[479]), .c(c[479]),
.sel(sel));
```

```
    xtmux3b i480(.xtout(xtout[480]), .a(a[480]), .b(b[480]), .c(c[480]),
.sel(sel));
    xtmux3b i481(.xtout(xtout[481]), .a(a[481]), .b(b[481]), .c(c[481]),
.sel(sel));
    xtmux3b i482(.xtout(xtout[482]), .a(a[482]), .b(b[482]), .c(c[482]),
.sel(sel));
    xtmux3b i483(.xtout(xtout[483]), .a(a[483]), .b(b[483]), .c(c[483]),
.sel(sel));
    xtmux3b i484(.xtout(xtout[484]), .a(a[484]), .b(b[484]), .c(c[484]),
.sel(sel));
    xtmux3b i485(.xtout(xtout[485]), .a(a[485]), .b(b[485]), .c(c[485]),
.sel(sel));
    xtmux3b i486(.xtout(xtout[486]), .a(a[486]), .b(b[486]), .c(c[486]),
.sel(sel));
    xtmux3b i487(.xtout(xtout[487]), .a(a[487]), .b(b[487]), .c(c[487]),
.sel(sel));
    xtmux3b i488(.xtout(xtout[488]), .a(a[488]), .b(b[488]), .c(c[488]),
.sel(sel));
    xtmux3b i489(.xtout(xtout[489]), .a(a[489]), .b(b[489]), .c(c[489]),
.sel(sel));
    xtmux3b i490(.xtout(xtout[490]), .a(a[490]), .b(b[490]), .c(c[490]),
.sel(sel));
    xtmux3b i491(.xtout(xtout[491]), .a(a[491]), .b(b[491]), .c(c[491]),
.sel(sel));
    xtmux3b i492(.xtout(xtout[492]), .a(a[492]), .b(b[492]), .c(c[492]),
.sel(sel));
    xtmux3b i493(.xtout(xtout[493]), .a(a[493]), .b(b[493]), .c(c[493]),
.sel(sel));
    xtmux3b i494(.xtout(xtout[494]), .a(a[494]), .b(b[494]), .c(c[494]),
.sel(sel));
    xtmux3b i495(.xtout(xtout[495]), .a(a[495]), .b(b[495]), .c(c[495]),
.sel(sel));
    xtmux3b i496(.xtout(xtout[496]), .a(a[496]), .b(b[496]), .c(c[496]),
.sel(sel));
    xtmux3b i497(.xtout(xtout[497]), .a(a[497]), .b(b[497]), .c(c[497]),
.sel(sel));
    xtmux3b i498(.xtout(xtout[498]), .a(a[498]), .b(b[498]), .c(c[498]),
.sel(sel));
    xtmux3b i499(.xtout(xtout[499]), .a(a[499]), .b(b[499]), .c(c[499]),
.sel(sel));
    xtmux3b i500(.xtout(xtout[500]), .a(a[500]), .b(b[500]), .c(c[500]),
.sel(sel));
    xtmux3b i501(.xtout(xtout[501]), .a(a[501]), .b(b[501]), .c(c[501]),
.sel(sel));
    xtmux3b i502(.xtout(xtout[502]), .a(a[502]), .b(b[502]), .c(c[502]),
.sel(sel));
    xtmux3b i503(.xtout(xtout[503]), .a(a[503]), .b(b[503]), .c(c[503]),
.sel(sel));
    xtmux3b i504(.xtout(xtout[504]), .a(a[504]), .b(b[504]), .c(c[504]),
.sel(sel));
    xtmux3b i505(.xtout(xtout[505]), .a(a[505]), .b(b[505]), .c(c[505]),
.sel(sel));
    xtmux3b i506(.xtout(xtout[506]), .a(a[506]), .b(b[506]), .c(c[506]),
.sel(sel));
    xtmux3b i507(.xtout(xtout[507]), .a(a[507]), .b(b[507]), .c(c[507]),
.sel(sel));
```

```
    xtmux3b i508(.xtout(xtout[508]), .a(a[508]), .b(b[508]), .c(c[508]),
.sel(sel));
    xtmux3b i509(.xtout(xtout[509]), .a(a[509]), .b(b[509]), .c(c[509]),
.sel(sel));
    xtmux3b i510(.xtout(xtout[510]), .a(a[510]), .b(b[510]), .c(c[510]),
.sel(sel));
    xtmux3b i511(.xtout(xtout[511]), .a(a[511]), .b(b[511]), .c(c[511]),
.sel(sel));
    xtmux3b i512(.xtout(xtout[512]), .a(a[512]), .b(b[512]), .c(c[512]),
.sel(sel));
    xtmux3b i513(.xtout(xtout[513]), .a(a[513]), .b(b[513]), .c(c[513]),
.sel(sel));
    xtmux3b i514(.xtout(xtout[514]), .a(a[514]), .b(b[514]), .c(c[514]),
.sel(sel));
    xtmux3b i515(.xtout(xtout[515]), .a(a[515]), .b(b[515]), .c(c[515]),
.sel(sel));
    xtmux3b i516(.xtout(xtout[516]), .a(a[516]), .b(b[516]), .c(c[516]),
.sel(sel));
    xtmux3b i517(.xtout(xtout[517]), .a(a[517]), .b(b[517]), .c(c[517]),
.sel(sel));
    xtmux3b i518(.xtout(xtout[518]), .a(a[518]), .b(b[518]), .c(c[518]),
.sel(sel));
    xtmux3b i519(.xtout(xtout[519]), .a(a[519]), .b(b[519]), .c(c[519]),
.sel(sel));
    xtmux3b i520(.xtout(xtout[520]), .a(a[520]), .b(b[520]), .c(c[520]),
.sel(sel));
    xtmux3b i521(.xtout(xtout[521]), .a(a[521]), .b(b[521]), .c(c[521]),
.sel(sel));
    xtmux3b i522(.xtout(xtout[522]), .a(a[522]), .b(b[522]), .c(c[522]),
.sel(sel));
    xtmux3b i523(.xtout(xtout[523]), .a(a[523]), .b(b[523]), .c(c[523]),
.sel(sel));
    xtmux3b i524(.xtout(xtout[524]), .a(a[524]), .b(b[524]), .c(c[524]),
.sel(sel));
    xtmux3b i525(.xtout(xtout[525]), .a(a[525]), .b(b[525]), .c(c[525]),
.sel(sel));
    xtmux3b i526(.xtout(xtout[526]), .a(a[526]), .b(b[526]), .c(c[526]),
.sel(sel));
    xtmux3b i527(.xtout(xtout[527]), .a(a[527]), .b(b[527]), .c(c[527]),
.sel(sel));
    xtmux3b i528(.xtout(xtout[528]), .a(a[528]), .b(b[528]), .c(c[528]),
.sel(sel));
    xtmux3b i529(.xtout(xtout[529]), .a(a[529]), .b(b[529]), .c(c[529]),
.sel(sel));
    xtmux3b i530(.xtout(xtout[530]), .a(a[530]), .b(b[530]), .c(c[530]),
.sel(sel));
    xtmux3b i531(.xtout(xtout[531]), .a(a[531]), .b(b[531]), .c(c[531]),
.sel(sel));
    xtmux3b i532(.xtout(xtout[532]), .a(a[532]), .b(b[532]), .c(c[532]),
.sel(sel));
    xtmux3b i533(.xtout(xtout[533]), .a(a[533]), .b(b[533]), .c(c[533]),
.sel(sel));
    xtmux3b i534(.xtout(xtout[534]), .a(a[534]), .b(b[534]), .c(c[534]),
.sel(sel));
    xtmux3b i535(.xtout(xtout[535]), .a(a[535]), .b(b[535]), .c(c[535]),
.sel(sel));
```

```
    xtmux3b i536(.xtout(xtout[536]), .a(a[536]), .b(b[536]), .c(c[536]),
.sel(sel));
    xtmux3b i537(.xtout(xtout[537]), .a(a[537]), .b(b[537]), .c(c[537]),
.sel(sel));
    xtmux3b i538(.xtout(xtout[538]), .a(a[538]), .b(b[538]), .c(c[538]),
.sel(sel));
    xtmux3b i539(.xtout(xtout[539]), .a(a[539]), .b(b[539]), .c(c[539]),
.sel(sel));
    xtmux3b i540(.xtout(xtout[540]), .a(a[540]), .b(b[540]), .c(c[540]),
.sel(sel));
    xtmux3b i541(.xtout(xtout[541]), .a(a[541]), .b(b[541]), .c(c[541]),
.sel(sel));
    xtmux3b i542(.xtout(xtout[542]), .a(a[542]), .b(b[542]), .c(c[542]),
.sel(sel));
    xtmux3b i543(.xtout(xtout[543]), .a(a[543]), .b(b[543]), .c(c[543]),
.sel(sel));
    xtmux3b i544(.xtout(xtout[544]), .a(a[544]), .b(b[544]), .c(c[544]),
.sel(sel));
    xtmux3b i545(.xtout(xtout[545]), .a(a[545]), .b(b[545]), .c(c[545]),
.sel(sel));
    xtmux3b i546(.xtout(xtout[546]), .a(a[546]), .b(b[546]), .c(c[546]),
.sel(sel));
    xtmux3b i547(.xtout(xtout[547]), .a(a[547]), .b(b[547]), .c(c[547]),
.sel(sel));
    xtmux3b i548(.xtout(xtout[548]), .a(a[548]), .b(b[548]), .c(c[548]),
.sel(sel));
    xtmux3b i549(.xtout(xtout[549]), .a(a[549]), .b(b[549]), .c(c[549]),
.sel(sel));
    xtmux3b i550(.xtout(xtout[550]), .a(a[550]), .b(b[550]), .c(c[550]),
.sel(sel));
    xtmux3b i551(.xtout(xtout[551]), .a(a[551]), .b(b[551]), .c(c[551]),
.sel(sel));
    xtmux3b i552(.xtout(xtout[552]), .a(a[552]), .b(b[552]), .c(c[552]),
.sel(sel));
    xtmux3b i553(.xtout(xtout[553]), .a(a[553]), .b(b[553]), .c(c[553]),
.sel(sel));
    xtmux3b i554(.xtout(xtout[554]), .a(a[554]), .b(b[554]), .c(c[554]),
.sel(sel));
    xtmux3b i555(.xtout(xtout[555]), .a(a[555]), .b(b[555]), .c(c[555]),
.sel(sel));
    xtmux3b i556(.xtout(xtout[556]), .a(a[556]), .b(b[556]), .c(c[556]),
.sel(sel));
    xtmux3b i557(.xtout(xtout[557]), .a(a[557]), .b(b[557]), .c(c[557]),
.sel(sel));
    xtmux3b i558(.xtout(xtout[558]), .a(a[558]), .b(b[558]), .c(c[558]),
.sel(sel));
    xtmux3b i559(.xtout(xtout[559]), .a(a[559]), .b(b[559]), .c(c[559]),
.sel(sel));
    xtmux3b i560(.xtout(xtout[560]), .a(a[560]), .b(b[560]), .c(c[560]),
.sel(sel));
    xtmux3b i561(.xtout(xtout[561]), .a(a[561]), .b(b[561]), .c(c[561]),
.sel(sel));
    xtmux3b i562(.xtout(xtout[562]), .a(a[562]), .b(b[562]), .c(c[562]),
.sel(sel));
    xtmux3b i563(.xtout(xtout[563]), .a(a[563]), .b(b[563]), .c(c[563]),
.sel(sel));
```

```
    xtmux3b i564(.xtout(xtout[564]), .a(a[564]), .b(b[564]), .c(c[564]),
.sel(sel));
    xtmux3b i565(.xtout(xtout[565]), .a(a[565]), .b(b[565]), .c(c[565]),
.sel(sel));
    xtmux3b i566(.xtout(xtout[566]), .a(a[566]), .b(b[566]), .c(c[566]),
.sel(sel));
    xtmux3b i567(.xtout(xtout[567]), .a(a[567]), .b(b[567]), .c(c[567]),
.sel(sel));
    xtmux3b i568(.xtout(xtout[568]), .a(a[568]), .b(b[568]), .c(c[568]),
.sel(sel));
    xtmux3b i569(.xtout(xtout[569]), .a(a[569]), .b(b[569]), .c(c[569]),
.sel(sel));
    xtmux3b i570(.xtout(xtout[570]), .a(a[570]), .b(b[570]), .c(c[570]),
.sel(sel));
    xtmux3b i571(.xtout(xtout[571]), .a(a[571]), .b(b[571]), .c(c[571]),
.sel(sel));
    xtmux3b i572(.xtout(xtout[572]), .a(a[572]), .b(b[572]), .c(c[572]),
.sel(sel));
    xtmux3b i573(.xtout(xtout[573]), .a(a[573]), .b(b[573]), .c(c[573]),
.sel(sel));
    xtmux3b i574(.xtout(xtout[574]), .a(a[574]), .b(b[574]), .c(c[574]),
.sel(sel));
    xtmux3b i575(.xtout(xtout[575]), .a(a[575]), .b(b[575]), .c(c[575]),
.sel(sel));
    xtmux3b i576(.xtout(xtout[576]), .a(a[576]), .b(b[576]), .c(c[576]),
.sel(sel));
    xtmux3b i577(.xtout(xtout[577]), .a(a[577]), .b(b[577]), .c(c[577]),
.sel(sel));
    xtmux3b i578(.xtout(xtout[578]), .a(a[578]), .b(b[578]), .c(c[578]),
.sel(sel));
    xtmux3b i579(.xtout(xtout[579]), .a(a[579]), .b(b[579]), .c(c[579]),
.sel(sel));
    xtmux3b i580(.xtout(xtout[580]), .a(a[580]), .b(b[580]), .c(c[580]),
.sel(sel));
    xtmux3b i581(.xtout(xtout[581]), .a(a[581]), .b(b[581]), .c(c[581]),
.sel(sel));
    xtmux3b i582(.xtout(xtout[582]), .a(a[582]), .b(b[582]), .c(c[582]),
.sel(sel));
    xtmux3b i583(.xtout(xtout[583]), .a(a[583]), .b(b[583]), .c(c[583]),
.sel(sel));
    xtmux3b i584(.xtout(xtout[584]), .a(a[584]), .b(b[584]), .c(c[584]),
.sel(sel));
    xtmux3b i585(.xtout(xtout[585]), .a(a[585]), .b(b[585]), .c(c[585]),
.sel(sel));
    xtmux3b i586(.xtout(xtout[586]), .a(a[586]), .b(b[586]), .c(c[586]),
.sel(sel));
    xtmux3b i587(.xtout(xtout[587]), .a(a[587]), .b(b[587]), .c(c[587]),
.sel(sel));
    xtmux3b i588(.xtout(xtout[588]), .a(a[588]), .b(b[588]), .c(c[588]),
.sel(sel));
    xtmux3b i589(.xtout(xtout[589]), .a(a[589]), .b(b[589]), .c(c[589]),
.sel(sel));
    xtmux3b i590(.xtout(xtout[590]), .a(a[590]), .b(b[590]), .c(c[590]),
.sel(sel));
    xtmux3b i591(.xtout(xtout[591]), .a(a[591]), .b(b[591]), .c(c[591]),
.sel(sel));
```

```
     xtmux3b i592(.xtout(xtout[592]), .a(a[592]), .b(b[592]), .c(c[592]),
.sel(sel));
     xtmux3b i593(.xtout(xtout[593]), .a(a[593]), .b(b[593]), .c(c[593]),
.sel(sel));
     xtmux3b i594(.xtout(xtout[594]), .a(a[594]), .b(b[594]), .c(c[594]),
.sel(sel));
     xtmux3b i595(.xtout(xtout[595]), .a(a[595]), .b(b[595]), .c(c[595]),
.sel(sel));
     xtmux3b i596(.xtout(xtout[596]), .a(a[596]), .b(b[596]), .c(c[596]),
.sel(sel));
     xtmux3b i597(.xtout(xtout[597]), .a(a[597]), .b(b[597]), .c(c[597]),
.sel(sel));
     xtmux3b i598(.xtout(xtout[598]), .a(a[598]), .b(b[598]), .c(c[598]),
.sel(sel));
     xtmux3b i599(.xtout(xtout[599]), .a(a[599]), .b(b[599]), .c(c[599]),
.sel(sel));
     xtmux3b i600(.xtout(xtout[600]), .a(a[600]), .b(b[600]), .c(c[600]),
.sel(sel));
     xtmux3b i601(.xtout(xtout[601]), .a(a[601]), .b(b[601]), .c(c[601]),
.sel(sel));
     xtmux3b i602(.xtout(xtout[602]), .a(a[602]), .b(b[602]), .c(c[602]),
.sel(sel));
     xtmux3b i603(.xtout(xtout[603]), .a(a[603]), .b(b[603]), .c(c[603]),
.sel(sel));
     xtmux3b i604(.xtout(xtout[604]), .a(a[604]), .b(b[604]), .c(c[604]),
.sel(sel));
     xtmux3b i605(.xtout(xtout[605]), .a(a[605]), .b(b[605]), .c(c[605]),
.sel(sel));
     xtmux3b i606(.xtout(xtout[606]), .a(a[606]), .b(b[606]), .c(c[606]),
.sel(sel));
     xtmux3b i607(.xtout(xtout[607]), .a(a[607]), .b(b[607]), .c(c[607]),
.sel(sel));
     xtmux3b i608(.xtout(xtout[608]), .a(a[608]), .b(b[608]), .c(c[608]),
.sel(sel));
     xtmux3b i609(.xtout(xtout[609]), .a(a[609]), .b(b[609]), .c(c[609]),
.sel(sel));
     xtmux3b i610(.xtout(xtout[610]), .a(a[610]), .b(b[610]), .c(c[610]),
.sel(sel));
     xtmux3b i611(.xtout(xtout[611]), .a(a[611]), .b(b[611]), .c(c[611]),
.sel(sel));
     xtmux3b i612(.xtout(xtout[612]), .a(a[612]), .b(b[612]), .c(c[612]),
.sel(sel));
     xtmux3b i613(.xtout(xtout[613]), .a(a[613]), .b(b[613]), .c(c[613]),
.sel(sel));
     xtmux3b i614(.xtout(xtout[614]), .a(a[614]), .b(b[614]), .c(c[614]),
.sel(sel));
     xtmux3b i615(.xtout(xtout[615]), .a(a[615]), .b(b[615]), .c(c[615]),
.sel(sel));
     xtmux3b i616(.xtout(xtout[616]), .a(a[616]), .b(b[616]), .c(c[616]),
.sel(sel));
     xtmux3b i617(.xtout(xtout[617]), .a(a[617]), .b(b[617]), .c(c[617]),
.sel(sel));
     xtmux3b i618(.xtout(xtout[618]), .a(a[618]), .b(b[618]), .c(c[618]),
.sel(sel));
     xtmux3b i619(.xtout(xtout[619]), .a(a[619]), .b(b[619]), .c(c[619]),
.sel(sel));
```

```
xtmux3b i620(.xtout(xtout[620]), .a(a[620]), .b(b[620]), .c(c[620]),
.sel(sel));
xtmux3b i621(.xtout(xtout[621]), .a(a[621]), .b(b[621]), .c(c[621]),
.sel(sel));
xtmux3b i622(.xtout(xtout[622]), .a(a[622]), .b(b[622]), .c(c[622]),
.sel(sel));
xtmux3b i623(.xtout(xtout[623]), .a(a[623]), .b(b[623]), .c(c[623]),
.sel(sel));
xtmux3b i624(.xtout(xtout[624]), .a(a[624]), .b(b[624]), .c(c[624]),
.sel(sel));
xtmux3b i625(.xtout(xtout[625]), .a(a[625]), .b(b[625]), .c(c[625]),
.sel(sel));
xtmux3b i626(.xtout(xtout[626]), .a(a[626]), .b(b[626]), .c(c[626]),
.sel(sel));
xtmux3b i627(.xtout(xtout[627]), .a(a[627]), .b(b[627]), .c(c[627]),
.sel(sel));
xtmux3b i628(.xtout(xtout[628]), .a(a[628]), .b(b[628]), .c(c[628]),
.sel(sel));
xtmux3b i629(.xtout(xtout[629]), .a(a[629]), .b(b[629]), .c(c[629]),
.sel(sel));
xtmux3b i630(.xtout(xtout[630]), .a(a[630]), .b(b[630]), .c(c[630]),
.sel(sel));
xtmux3b i631(.xtout(xtout[631]), .a(a[631]), .b(b[631]), .c(c[631]),
.sel(sel));
xtmux3b i632(.xtout(xtout[632]), .a(a[632]), .b(b[632]), .c(c[632]),
.sel(sel));
xtmux3b i633(.xtout(xtout[633]), .a(a[633]), .b(b[633]), .c(c[633]),
.sel(sel));
xtmux3b i634(.xtout(xtout[634]), .a(a[634]), .b(b[634]), .c(c[634]),
.sel(sel));
xtmux3b i635(.xtout(xtout[635]), .a(a[635]), .b(b[635]), .c(c[635]),
.sel(sel));
xtmux3b i636(.xtout(xtout[636]), .a(a[636]), .b(b[636]), .c(c[636]),
.sel(sel));
xtmux3b i637(.xtout(xtout[637]), .a(a[637]), .b(b[637]), .c(c[637]),
.sel(sel));
xtmux3b i638(.xtout(xtout[638]), .a(a[638]), .b(b[638]), .c(c[638]),
.sel(sel));
xtmux3b i639(.xtout(xtout[639]), .a(a[639]), .b(b[639]), .c(c[639]),
.sel(sel));
xtmux3b i640(.xtout(xtout[640]), .a(a[640]), .b(b[640]), .c(c[640]),
.sel(sel));
xtmux3b i641(.xtout(xtout[641]), .a(a[641]), .b(b[641]), .c(c[641]),
.sel(sel));
xtmux3b i642(.xtout(xtout[642]), .a(a[642]), .b(b[642]), .c(c[642]),
.sel(sel));
xtmux3b i643(.xtout(xtout[643]), .a(a[643]), .b(b[643]), .c(c[643]),
.sel(sel));
xtmux3b i644(.xtout(xtout[644]), .a(a[644]), .b(b[644]), .c(c[644]),
.sel(sel));
xtmux3b i645(.xtout(xtout[645]), .a(a[645]), .b(b[645]), .c(c[645]),
.sel(sel));
xtmux3b i646(.xtout(xtout[646]), .a(a[646]), .b(b[646]), .c(c[646]),
.sel(sel));
xtmux3b i647(.xtout(xtout[647]), .a(a[647]), .b(b[647]), .c(c[647]),
.sel(sel));
```

```
xtmux3b i648(.xtout(xtout[648]), .a(a[648]), .b(b[648]), .c(c[648]),
.sel(sel));
xtmux3b i649(.xtout(xtout[649]), .a(a[649]), .b(b[649]), .c(c[649]),
.sel(sel));
xtmux3b i650(.xtout(xtout[650]), .a(a[650]), .b(b[650]), .c(c[650]),
.sel(sel));
xtmux3b i651(.xtout(xtout[651]), .a(a[651]), .b(b[651]), .c(c[651]),
.sel(sel));
xtmux3b i652(.xtout(xtout[652]), .a(a[652]), .b(b[652]), .c(c[652]),
.sel(sel));
xtmux3b i653(.xtout(xtout[653]), .a(a[653]), .b(b[653]), .c(c[653]),
.sel(sel));
xtmux3b i654(.xtout(xtout[654]), .a(a[654]), .b(b[654]), .c(c[654]),
.sel(sel));
xtmux3b i655(.xtout(xtout[655]), .a(a[655]), .b(b[655]), .c(c[655]),
.sel(sel));
xtmux3b i656(.xtout(xtout[656]), .a(a[656]), .b(b[656]), .c(c[656]),
.sel(sel));
xtmux3b i657(.xtout(xtout[657]), .a(a[657]), .b(b[657]), .c(c[657]),
.sel(sel));
xtmux3b i658(.xtout(xtout[658]), .a(a[658]), .b(b[658]), .c(c[658]),
.sel(sel));
xtmux3b i659(.xtout(xtout[659]), .a(a[659]), .b(b[659]), .c(c[659]),
.sel(sel));
xtmux3b i660(.xtout(xtout[660]), .a(a[660]), .b(b[660]), .c(c[660]),
.sel(sel));
xtmux3b i661(.xtout(xtout[661]), .a(a[661]), .b(b[661]), .c(c[661]),
.sel(sel));
xtmux3b i662(.xtout(xtout[662]), .a(a[662]), .b(b[662]), .c(c[662]),
.sel(sel));
xtmux3b i663(.xtout(xtout[663]), .a(a[663]), .b(b[663]), .c(c[663]),
.sel(sel));
xtmux3b i664(.xtout(xtout[664]), .a(a[664]), .b(b[664]), .c(c[664]),
.sel(sel));
xtmux3b i665(.xtout(xtout[665]), .a(a[665]), .b(b[665]), .c(c[665]),
.sel(sel));
xtmux3b i666(.xtout(xtout[666]), .a(a[666]), .b(b[666]), .c(c[666]),
.sel(sel));
xtmux3b i667(.xtout(xtout[667]), .a(a[667]), .b(b[667]), .c(c[667]),
.sel(sel));
xtmux3b i668(.xtout(xtout[668]), .a(a[668]), .b(b[668]), .c(c[668]),
.sel(sel));
xtmux3b i669(.xtout(xtout[669]), .a(a[669]), .b(b[669]), .c(c[669]),
.sel(sel));
xtmux3b i670(.xtout(xtout[670]), .a(a[670]), .b(b[670]), .c(c[670]),
.sel(sel));
xtmux3b i671(.xtout(xtout[671]), .a(a[671]), .b(b[671]), .c(c[671]),
.sel(sel));
xtmux3b i672(.xtout(xtout[672]), .a(a[672]), .b(b[672]), .c(c[672]),
.sel(sel));
xtmux3b i673(.xtout(xtout[673]), .a(a[673]), .b(b[673]), .c(c[673]),
.sel(sel));
xtmux3b i674(.xtout(xtout[674]), .a(a[674]), .b(b[674]), .c(c[674]),
.sel(sel));
xtmux3b i675(.xtout(xtout[675]), .a(a[675]), .b(b[675]), .c(c[675]),
.sel(sel));
```

```
    xtmux3b i676(.xtout(xtout[676]), .a(a[676]), .b(b[676]), .c(c[676]),
.sel(sel));
    xtmux3b i677(.xtout(xtout[677]), .a(a[677]), .b(b[677]), .c(c[677]),
.sel(sel));
    xtmux3b i678(.xtout(xtout[678]), .a(a[678]), .b(b[678]), .c(c[678]),
.sel(sel));
    xtmux3b i679(.xtout(xtout[679]), .a(a[679]), .b(b[679]), .c(c[679]),
.sel(sel));
    xtmux3b i680(.xtout(xtout[680]), .a(a[680]), .b(b[680]), .c(c[680]),
.sel(sel));
    xtmux3b i681(.xtout(xtout[681]), .a(a[681]), .b(b[681]), .c(c[681]),
.sel(sel));
    xtmux3b i682(.xtout(xtout[682]), .a(a[682]), .b(b[682]), .c(c[682]),
.sel(sel));
    xtmux3b i683(.xtout(xtout[683]), .a(a[683]), .b(b[683]), .c(c[683]),
.sel(sel));
    xtmux3b i684(.xtout(xtout[684]), .a(a[684]), .b(b[684]), .c(c[684]),
.sel(sel));
    xtmux3b i685(.xtout(xtout[685]), .a(a[685]), .b(b[685]), .c(c[685]),
.sel(sel));
    xtmux3b i686(.xtout(xtout[686]), .a(a[686]), .b(b[686]), .c(c[686]),
.sel(sel));
    xtmux3b i687(.xtout(xtout[687]), .a(a[687]), .b(b[687]), .c(c[687]),
.sel(sel));
    xtmux3b i688(.xtout(xtout[688]), .a(a[688]), .b(b[688]), .c(c[688]),
.sel(sel));
    xtmux3b i689(.xtout(xtout[689]), .a(a[689]), .b(b[689]), .c(c[689]),
.sel(sel));
    xtmux3b i690(.xtout(xtout[690]), .a(a[690]), .b(b[690]), .c(c[690]),
.sel(sel));
    xtmux3b i691(.xtout(xtout[691]), .a(a[691]), .b(b[691]), .c(c[691]),
.sel(sel));
    xtmux3b i692(.xtout(xtout[692]), .a(a[692]), .b(b[692]), .c(c[692]),
.sel(sel));
    xtmux3b i693(.xtout(xtout[693]), .a(a[693]), .b(b[693]), .c(c[693]),
.sel(sel));
    xtmux3b i694(.xtout(xtout[694]), .a(a[694]), .b(b[694]), .c(c[694]),
.sel(sel));
    xtmux3b i695(.xtout(xtout[695]), .a(a[695]), .b(b[695]), .c(c[695]),
.sel(sel));
    xtmux3b i696(.xtout(xtout[696]), .a(a[696]), .b(b[696]), .c(c[696]),
.sel(sel));
    xtmux3b i697(.xtout(xtout[697]), .a(a[697]), .b(b[697]), .c(c[697]),
.sel(sel));
    xtmux3b i698(.xtout(xtout[698]), .a(a[698]), .b(b[698]), .c(c[698]),
.sel(sel));
    xtmux3b i699(.xtout(xtout[699]), .a(a[699]), .b(b[699]), .c(c[699]),
.sel(sel));
    xtmux3b i700(.xtout(xtout[700]), .a(a[700]), .b(b[700]), .c(c[700]),
.sel(sel));
    xtmux3b i701(.xtout(xtout[701]), .a(a[701]), .b(b[701]), .c(c[701]),
.sel(sel));
    xtmux3b i702(.xtout(xtout[702]), .a(a[702]), .b(b[702]), .c(c[702]),
.sel(sel));
    xtmux3b i703(.xtout(xtout[703]), .a(a[703]), .b(b[703]), .c(c[703]),
.sel(sel));
```

```
    xtmux3b i704(.xtout(xtout[704]), .a(a[704]), .b(b[704]), .c(c[704]),
.sel(sel));
    xtmux3b i705(.xtout(xtout[705]), .a(a[705]), .b(b[705]), .c(c[705]),
.sel(sel));
    xtmux3b i706(.xtout(xtout[706]), .a(a[706]), .b(b[706]), .c(c[706]),
.sel(sel));
    xtmux3b i707(.xtout(xtout[707]), .a(a[707]), .b(b[707]), .c(c[707]),
.sel(sel));
    xtmux3b i708(.xtout(xtout[708]), .a(a[708]), .b(b[708]), .c(c[708]),
.sel(sel));
    xtmux3b i709(.xtout(xtout[709]), .a(a[709]), .b(b[709]), .c(c[709]),
.sel(sel));
    xtmux3b i710(.xtout(xtout[710]), .a(a[710]), .b(b[710]), .c(c[710]),
.sel(sel));
    xtmux3b i711(.xtout(xtout[711]), .a(a[711]), .b(b[711]), .c(c[711]),
.sel(sel));
    xtmux3b i712(.xtout(xtout[712]), .a(a[712]), .b(b[712]), .c(c[712]),
.sel(sel));
    xtmux3b i713(.xtout(xtout[713]), .a(a[713]), .b(b[713]), .c(c[713]),
.sel(sel));
    xtmux3b i714(.xtout(xtout[714]), .a(a[714]), .b(b[714]), .c(c[714]),
.sel(sel));
    xtmux3b i715(.xtout(xtout[715]), .a(a[715]), .b(b[715]), .c(c[715]),
.sel(sel));
    xtmux3b i716(.xtout(xtout[716]), .a(a[716]), .b(b[716]), .c(c[716]),
.sel(sel));
    xtmux3b i717(.xtout(xtout[717]), .a(a[717]), .b(b[717]), .c(c[717]),
.sel(sel));
    xtmux3b i718(.xtout(xtout[718]), .a(a[718]), .b(b[718]), .c(c[718]),
.sel(sel));
    xtmux3b i719(.xtout(xtout[719]), .a(a[719]), .b(b[719]), .c(c[719]),
.sel(sel));
    xtmux3b i720(.xtout(xtout[720]), .a(a[720]), .b(b[720]), .c(c[720]),
.sel(sel));
    xtmux3b i721(.xtout(xtout[721]), .a(a[721]), .b(b[721]), .c(c[721]),
.sel(sel));
    xtmux3b i722(.xtout(xtout[722]), .a(a[722]), .b(b[722]), .c(c[722]),
.sel(sel));
    xtmux3b i723(.xtout(xtout[723]), .a(a[723]), .b(b[723]), .c(c[723]),
.sel(sel));
    xtmux3b i724(.xtout(xtout[724]), .a(a[724]), .b(b[724]), .c(c[724]),
.sel(sel));
    xtmux3b i725(.xtout(xtout[725]), .a(a[725]), .b(b[725]), .c(c[725]),
.sel(sel));
    xtmux3b i726(.xtout(xtout[726]), .a(a[726]), .b(b[726]), .c(c[726]),
.sel(sel));
    xtmux3b i727(.xtout(xtout[727]), .a(a[727]), .b(b[727]), .c(c[727]),
.sel(sel));
    xtmux3b i728(.xtout(xtout[728]), .a(a[728]), .b(b[728]), .c(c[728]),
.sel(sel));
    xtmux3b i729(.xtout(xtout[729]), .a(a[729]), .b(b[729]), .c(c[729]),
.sel(sel));
    xtmux3b i730(.xtout(xtout[730]), .a(a[730]), .b(b[730]), .c(c[730]),
.sel(sel));
    xtmux3b i731(.xtout(xtout[731]), .a(a[731]), .b(b[731]), .c(c[731]),
.sel(sel));
```

```
    xtmux3b i732(.xtout(xtout[732]), .a(a[732]), .b(b[732]), .c(c[732]),
.sel(sel));
    xtmux3b i733(.xtout(xtout[733]), .a(a[733]), .b(b[733]), .c(c[733]),
.sel(sel));
    xtmux3b i734(.xtout(xtout[734]), .a(a[734]), .b(b[734]), .c(c[734]),
.sel(sel));
    xtmux3b i735(.xtout(xtout[735]), .a(a[735]), .b(b[735]), .c(c[735]),
.sel(sel));
    xtmux3b i736(.xtout(xtout[736]), .a(a[736]), .b(b[736]), .c(c[736]),
.sel(sel));
    xtmux3b i737(.xtout(xtout[737]), .a(a[737]), .b(b[737]), .c(c[737]),
.sel(sel));
    xtmux3b i738(.xtout(xtout[738]), .a(a[738]), .b(b[738]), .c(c[738]),
.sel(sel));
    xtmux3b i739(.xtout(xtout[739]), .a(a[739]), .b(b[739]), .c(c[739]),
.sel(sel));
    xtmux3b i740(.xtout(xtout[740]), .a(a[740]), .b(b[740]), .c(c[740]),
.sel(sel));
    xtmux3b i741(.xtout(xtout[741]), .a(a[741]), .b(b[741]), .c(c[741]),
.sel(sel));
    xtmux3b i742(.xtout(xtout[742]), .a(a[742]), .b(b[742]), .c(c[742]),
.sel(sel));
    xtmux3b i743(.xtout(xtout[743]), .a(a[743]), .b(b[743]), .c(c[743]),
.sel(sel));
    xtmux3b i744(.xtout(xtout[744]), .a(a[744]), .b(b[744]), .c(c[744]),
.sel(sel));
    xtmux3b i745(.xtout(xtout[745]), .a(a[745]), .b(b[745]), .c(c[745]),
.sel(sel));
    xtmux3b i746(.xtout(xtout[746]), .a(a[746]), .b(b[746]), .c(c[746]),
.sel(sel));
    xtmux3b i747(.xtout(xtout[747]), .a(a[747]), .b(b[747]), .c(c[747]),
.sel(sel));
    xtmux3b i748(.xtout(xtout[748]), .a(a[748]), .b(b[748]), .c(c[748]),
.sel(sel));
    xtmux3b i749(.xtout(xtout[749]), .a(a[749]), .b(b[749]), .c(c[749]),
.sel(sel));
    xtmux3b i750(.xtout(xtout[750]), .a(a[750]), .b(b[750]), .c(c[750]),
.sel(sel));
    xtmux3b i751(.xtout(xtout[751]), .a(a[751]), .b(b[751]), .c(c[751]),
.sel(sel));
    xtmux3b i752(.xtout(xtout[752]), .a(a[752]), .b(b[752]), .c(c[752]),
.sel(sel));
    xtmux3b i753(.xtout(xtout[753]), .a(a[753]), .b(b[753]), .c(c[753]),
.sel(sel));
    xtmux3b i754(.xtout(xtout[754]), .a(a[754]), .b(b[754]), .c(c[754]),
.sel(sel));
    xtmux3b i755(.xtout(xtout[755]), .a(a[755]), .b(b[755]), .c(c[755]),
.sel(sel));
    xtmux3b i756(.xtout(xtout[756]), .a(a[756]), .b(b[756]), .c(c[756]),
.sel(sel));
    xtmux3b i757(.xtout(xtout[757]), .a(a[757]), .b(b[757]), .c(c[757]),
.sel(sel));
    xtmux3b i758(.xtout(xtout[758]), .a(a[758]), .b(b[758]), .c(c[758]),
.sel(sel));
    xtmux3b i759(.xtout(xtout[759]), .a(a[759]), .b(b[759]), .c(c[759]),
.sel(sel));
```

```
    xtmux3b i760(.xtout(xtout[760]), .a(a[760]), .b(b[760]), .c(c[760]),
.sel(sel));
    xtmux3b i761(.xtout(xtout[761]), .a(a[761]), .b(b[761]), .c(c[761]),
.sel(sel));
    xtmux3b i762(.xtout(xtout[762]), .a(a[762]), .b(b[762]), .c(c[762]),
.sel(sel));
    xtmux3b i763(.xtout(xtout[763]), .a(a[763]), .b(b[763]), .c(c[763]),
.sel(sel));
    xtmux3b i764(.xtout(xtout[764]), .a(a[764]), .b(b[764]), .c(c[764]),
.sel(sel));
    xtmux3b i765(.xtout(xtout[765]), .a(a[765]), .b(b[765]), .c(c[765]),
.sel(sel));
    xtmux3b i766(.xtout(xtout[766]), .a(a[766]), .b(b[766]), .c(c[766]),
.sel(sel));
    xtmux3b i767(.xtout(xtout[767]), .a(a[767]), .b(b[767]), .c(c[767]),
.sel(sel));
    xtmux3b i768(.xtout(xtout[768]), .a(a[768]), .b(b[768]), .c(c[768]),
.sel(sel));
    xtmux3b i769(.xtout(xtout[769]), .a(a[769]), .b(b[769]), .c(c[769]),
.sel(sel));
    xtmux3b i770(.xtout(xtout[770]), .a(a[770]), .b(b[770]), .c(c[770]),
.sel(sel));
    xtmux3b i771(.xtout(xtout[771]), .a(a[771]), .b(b[771]), .c(c[771]),
.sel(sel));
    xtmux3b i772(.xtout(xtout[772]), .a(a[772]), .b(b[772]), .c(c[772]),
.sel(sel));
    xtmux3b i773(.xtout(xtout[773]), .a(a[773]), .b(b[773]), .c(c[773]),
.sel(sel));
    xtmux3b i774(.xtout(xtout[774]), .a(a[774]), .b(b[774]), .c(c[774]),
.sel(sel));
    xtmux3b i775(.xtout(xtout[775]), .a(a[775]), .b(b[775]), .c(c[775]),
.sel(sel));
    xtmux3b i776(.xtout(xtout[776]), .a(a[776]), .b(b[776]), .c(c[776]),
.sel(sel));
    xtmux3b i777(.xtout(xtout[777]), .a(a[777]), .b(b[777]), .c(c[777]),
.sel(sel));
    xtmux3b i778(.xtout(xtout[778]), .a(a[778]), .b(b[778]), .c(c[778]),
.sel(sel));
    xtmux3b i779(.xtout(xtout[779]), .a(a[779]), .b(b[779]), .c(c[779]),
.sel(sel));
    xtmux3b i780(.xtout(xtout[780]), .a(a[780]), .b(b[780]), .c(c[780]),
.sel(sel));
    xtmux3b i781(.xtout(xtout[781]), .a(a[781]), .b(b[781]), .c(c[781]),
.sel(sel));
    xtmux3b i782(.xtout(xtout[782]), .a(a[782]), .b(b[782]), .c(c[782]),
.sel(sel));
    xtmux3b i783(.xtout(xtout[783]), .a(a[783]), .b(b[783]), .c(c[783]),
.sel(sel));
    xtmux3b i784(.xtout(xtout[784]), .a(a[784]), .b(b[784]), .c(c[784]),
.sel(sel));
    xtmux3b i785(.xtout(xtout[785]), .a(a[785]), .b(b[785]), .c(c[785]),
.sel(sel));
    xtmux3b i786(.xtout(xtout[786]), .a(a[786]), .b(b[786]), .c(c[786]),
.sel(sel));
    xtmux3b i787(.xtout(xtout[787]), .a(a[787]), .b(b[787]), .c(c[787]),
.sel(sel));
```

```
    xtmux3b i788(.xtout(xtout[788]), .a(a[788]), .b(b[788]), .c(c[788]),
.sel(sel));
    xtmux3b i789(.xtout(xtout[789]), .a(a[789]), .b(b[789]), .c(c[789]),
.sel(sel));
    xtmux3b i790(.xtout(xtout[790]), .a(a[790]), .b(b[790]), .c(c[790]),
.sel(sel));
    xtmux3b i791(.xtout(xtout[791]), .a(a[791]), .b(b[791]), .c(c[791]),
.sel(sel));
    xtmux3b i792(.xtout(xtout[792]), .a(a[792]), .b(b[792]), .c(c[792]),
.sel(sel));
    xtmux3b i793(.xtout(xtout[793]), .a(a[793]), .b(b[793]), .c(c[793]),
.sel(sel));
    xtmux3b i794(.xtout(xtout[794]), .a(a[794]), .b(b[794]), .c(c[794]),
.sel(sel));
    xtmux3b i795(.xtout(xtout[795]), .a(a[795]), .b(b[795]), .c(c[795]),
.sel(sel));
    xtmux3b i796(.xtout(xtout[796]), .a(a[796]), .b(b[796]), .c(c[796]),
.sel(sel));
    xtmux3b i797(.xtout(xtout[797]), .a(a[797]), .b(b[797]), .c(c[797]),
.sel(sel));
    xtmux3b i798(.xtout(xtout[798]), .a(a[798]), .b(b[798]), .c(c[798]),
.sel(sel));
    xtmux3b i799(.xtout(xtout[799]), .a(a[799]), .b(b[799]), .c(c[799]),
.sel(sel));
    xtmux3b i800(.xtout(xtout[800]), .a(a[800]), .b(b[800]), .c(c[800]),
.sel(sel));
    xtmux3b i801(.xtout(xtout[801]), .a(a[801]), .b(b[801]), .c(c[801]),
.sel(sel));
    xtmux3b i802(.xtout(xtout[802]), .a(a[802]), .b(b[802]), .c(c[802]),
.sel(sel));
    xtmux3b i803(.xtout(xtout[803]), .a(a[803]), .b(b[803]), .c(c[803]),
.sel(sel));
    xtmux3b i804(.xtout(xtout[804]), .a(a[804]), .b(b[804]), .c(c[804]),
.sel(sel));
    xtmux3b i805(.xtout(xtout[805]), .a(a[805]), .b(b[805]), .c(c[805]),
.sel(sel));
    xtmux3b i806(.xtout(xtout[806]), .a(a[806]), .b(b[806]), .c(c[806]),
.sel(sel));
    xtmux3b i807(.xtout(xtout[807]), .a(a[807]), .b(b[807]), .c(c[807]),
.sel(sel));
    xtmux3b i808(.xtout(xtout[808]), .a(a[808]), .b(b[808]), .c(c[808]),
.sel(sel));
    xtmux3b i809(.xtout(xtout[809]), .a(a[809]), .b(b[809]), .c(c[809]),
.sel(sel));
    xtmux3b i810(.xtout(xtout[810]), .a(a[810]), .b(b[810]), .c(c[810]),
.sel(sel));
    xtmux3b i811(.xtout(xtout[811]), .a(a[811]), .b(b[811]), .c(c[811]),
.sel(sel));
    xtmux3b i812(.xtout(xtout[812]), .a(a[812]), .b(b[812]), .c(c[812]),
.sel(sel));
    xtmux3b i813(.xtout(xtout[813]), .a(a[813]), .b(b[813]), .c(c[813]),
.sel(sel));
    xtmux3b i814(.xtout(xtout[814]), .a(a[814]), .b(b[814]), .c(c[814]),
.sel(sel));
    xtmux3b i815(.xtout(xtout[815]), .a(a[815]), .b(b[815]), .c(c[815]),
.sel(sel));
```

```
    xtmux3b i816(.xtout(xtout[816]), .a(a[816]), .b(b[816]), .c(c[816]),
.sel(sel));
    xtmux3b i817(.xtout(xtout[817]), .a(a[817]), .b(b[817]), .c(c[817]),
.sel(sel));
    xtmux3b i818(.xtout(xtout[818]), .a(a[818]), .b(b[818]), .c(c[818]),
.sel(sel));
    xtmux3b i819(.xtout(xtout[819]), .a(a[819]), .b(b[819]), .c(c[819]),
.sel(sel));
    xtmux3b i820(.xtout(xtout[820]), .a(a[820]), .b(b[820]), .c(c[820]),
.sel(sel));
    xtmux3b i821(.xtout(xtout[821]), .a(a[821]), .b(b[821]), .c(c[821]),
.sel(sel));
    xtmux3b i822(.xtout(xtout[822]), .a(a[822]), .b(b[822]), .c(c[822]),
.sel(sel));
    xtmux3b i823(.xtout(xtout[823]), .a(a[823]), .b(b[823]), .c(c[823]),
.sel(sel));
    xtmux3b i824(.xtout(xtout[824]), .a(a[824]), .b(b[824]), .c(c[824]),
.sel(sel));
    xtmux3b i825(.xtout(xtout[825]), .a(a[825]), .b(b[825]), .c(c[825]),
.sel(sel));
    xtmux3b i826(.xtout(xtout[826]), .a(a[826]), .b(b[826]), .c(c[826]),
.sel(sel));
    xtmux3b i827(.xtout(xtout[827]), .a(a[827]), .b(b[827]), .c(c[827]),
.sel(sel));
    xtmux3b i828(.xtout(xtout[828]), .a(a[828]), .b(b[828]), .c(c[828]),
.sel(sel));
    xtmux3b i829(.xtout(xtout[829]), .a(a[829]), .b(b[829]), .c(c[829]),
.sel(sel));
    xtmux3b i830(.xtout(xtout[830]), .a(a[830]), .b(b[830]), .c(c[830]),
.sel(sel));
    xtmux3b i831(.xtout(xtout[831]), .a(a[831]), .b(b[831]), .c(c[831]),
.sel(sel));
    xtmux3b i832(.xtout(xtout[832]), .a(a[832]), .b(b[832]), .c(c[832]),
.sel(sel));
    xtmux3b i833(.xtout(xtout[833]), .a(a[833]), .b(b[833]), .c(c[833]),
.sel(sel));
    xtmux3b i834(.xtout(xtout[834]), .a(a[834]), .b(b[834]), .c(c[834]),
.sel(sel));
    xtmux3b i835(.xtout(xtout[835]), .a(a[835]), .b(b[835]), .c(c[835]),
.sel(sel));
    xtmux3b i836(.xtout(xtout[836]), .a(a[836]), .b(b[836]), .c(c[836]),
.sel(sel));
    xtmux3b i837(.xtout(xtout[837]), .a(a[837]), .b(b[837]), .c(c[837]),
.sel(sel));
    xtmux3b i838(.xtout(xtout[838]), .a(a[838]), .b(b[838]), .c(c[838]),
.sel(sel));
    xtmux3b i839(.xtout(xtout[839]), .a(a[839]), .b(b[839]), .c(c[839]),
.sel(sel));
    xtmux3b i840(.xtout(xtout[840]), .a(a[840]), .b(b[840]), .c(c[840]),
.sel(sel));
    xtmux3b i841(.xtout(xtout[841]), .a(a[841]), .b(b[841]), .c(c[841]),
.sel(sel));
    xtmux3b i842(.xtout(xtout[842]), .a(a[842]), .b(b[842]), .c(c[842]),
.sel(sel));
    xtmux3b i843(.xtout(xtout[843]), .a(a[843]), .b(b[843]), .c(c[843]),
.sel(sel));
```

```
    xtmux3b i844(.xtout(xtout[844]), .a(a[844]), .b(b[844]), .c(c[844]),
.sel(sel));
    xtmux3b i845(.xtout(xtout[845]), .a(a[845]), .b(b[845]), .c(c[845]),
.sel(sel));
    xtmux3b i846(.xtout(xtout[846]), .a(a[846]), .b(b[846]), .c(c[846]),
.sel(sel));
    xtmux3b i847(.xtout(xtout[847]), .a(a[847]), .b(b[847]), .c(c[847]),
.sel(sel));
    xtmux3b i848(.xtout(xtout[848]), .a(a[848]), .b(b[848]), .c(c[848]),
.sel(sel));
    xtmux3b i849(.xtout(xtout[849]), .a(a[849]), .b(b[849]), .c(c[849]),
.sel(sel));
    xtmux3b i850(.xtout(xtout[850]), .a(a[850]), .b(b[850]), .c(c[850]),
.sel(sel));
    xtmux3b i851(.xtout(xtout[851]), .a(a[851]), .b(b[851]), .c(c[851]),
.sel(sel));
    xtmux3b i852(.xtout(xtout[852]), .a(a[852]), .b(b[852]), .c(c[852]),
.sel(sel));
    xtmux3b i853(.xtout(xtout[853]), .a(a[853]), .b(b[853]), .c(c[853]),
.sel(sel));
    xtmux3b i854(.xtout(xtout[854]), .a(a[854]), .b(b[854]), .c(c[854]),
.sel(sel));
    xtmux3b i855(.xtout(xtout[855]), .a(a[855]), .b(b[855]), .c(c[855]),
.sel(sel));
    xtmux3b i856(.xtout(xtout[856]), .a(a[856]), .b(b[856]), .c(c[856]),
.sel(sel));
    xtmux3b i857(.xtout(xtout[857]), .a(a[857]), .b(b[857]), .c(c[857]),
.sel(sel));
    xtmux3b i858(.xtout(xtout[858]), .a(a[858]), .b(b[858]), .c(c[858]),
.sel(sel));
    xtmux3b i859(.xtout(xtout[859]), .a(a[859]), .b(b[859]), .c(c[859]),
.sel(sel));
    xtmux3b i860(.xtout(xtout[860]), .a(a[860]), .b(b[860]), .c(c[860]),
.sel(sel));
    xtmux3b i861(.xtout(xtout[861]), .a(a[861]), .b(b[861]), .c(c[861]),
.sel(sel));
    xtmux3b i862(.xtout(xtout[862]), .a(a[862]), .b(b[862]), .c(c[862]),
.sel(sel));
    xtmux3b i863(.xtout(xtout[863]), .a(a[863]), .b(b[863]), .c(c[863]),
.sel(sel));
    xtmux3b i864(.xtout(xtout[864]), .a(a[864]), .b(b[864]), .c(c[864]),
.sel(sel));
    xtmux3b i865(.xtout(xtout[865]), .a(a[865]), .b(b[865]), .c(c[865]),
.sel(sel));
    xtmux3b i866(.xtout(xtout[866]), .a(a[866]), .b(b[866]), .c(c[866]),
.sel(sel));
    xtmux3b i867(.xtout(xtout[867]), .a(a[867]), .b(b[867]), .c(c[867]),
.sel(sel));
    xtmux3b i868(.xtout(xtout[868]), .a(a[868]), .b(b[868]), .c(c[868]),
.sel(sel));
    xtmux3b i869(.xtout(xtout[869]), .a(a[869]), .b(b[869]), .c(c[869]),
.sel(sel));
    xtmux3b i870(.xtout(xtout[870]), .a(a[870]), .b(b[870]), .c(c[870]),
.sel(sel));
    xtmux3b i871(.xtout(xtout[871]), .a(a[871]), .b(b[871]), .c(c[871]),
.sel(sel));
```

```
    xtmux3b i872(.xtout(xtout[872]), .a(a[872]), .b(b[872]), .c(c[872]),
.sel(sel));
    xtmux3b i873(.xtout(xtout[873]), .a(a[873]), .b(b[873]), .c(c[873]),
.sel(sel));
    xtmux3b i874(.xtout(xtout[874]), .a(a[874]), .b(b[874]), .c(c[874]),
.sel(sel));
    xtmux3b i875(.xtout(xtout[875]), .a(a[875]), .b(b[875]), .c(c[875]),
.sel(sel));
    xtmux3b i876(.xtout(xtout[876]), .a(a[876]), .b(b[876]), .c(c[876]),
.sel(sel));
    xtmux3b i877(.xtout(xtout[877]), .a(a[877]), .b(b[877]), .c(c[877]),
.sel(sel));
    xtmux3b i878(.xtout(xtout[878]), .a(a[878]), .b(b[878]), .c(c[878]),
.sel(sel));
    xtmux3b i879(.xtout(xtout[879]), .a(a[879]), .b(b[879]), .c(c[879]),
.sel(sel));
    xtmux3b i880(.xtout(xtout[880]), .a(a[880]), .b(b[880]), .c(c[880]),
.sel(sel));
    xtmux3b i881(.xtout(xtout[881]), .a(a[881]), .b(b[881]), .c(c[881]),
.sel(sel));
    xtmux3b i882(.xtout(xtout[882]), .a(a[882]), .b(b[882]), .c(c[882]),
.sel(sel));
    xtmux3b i883(.xtout(xtout[883]), .a(a[883]), .b(b[883]), .c(c[883]),
.sel(sel));
    xtmux3b i884(.xtout(xtout[884]), .a(a[884]), .b(b[884]), .c(c[884]),
.sel(sel));
    xtmux3b i885(.xtout(xtout[885]), .a(a[885]), .b(b[885]), .c(c[885]),
.sel(sel));
    xtmux3b i886(.xtout(xtout[886]), .a(a[886]), .b(b[886]), .c(c[886]),
.sel(sel));
    xtmux3b i887(.xtout(xtout[887]), .a(a[887]), .b(b[887]), .c(c[887]),
.sel(sel));
    xtmux3b i888(.xtout(xtout[888]), .a(a[888]), .b(b[888]), .c(c[888]),
.sel(sel));
    xtmux3b i889(.xtout(xtout[889]), .a(a[889]), .b(b[889]), .c(c[889]),
.sel(sel));
    xtmux3b i890(.xtout(xtout[890]), .a(a[890]), .b(b[890]), .c(c[890]),
.sel(sel));
    xtmux3b i891(.xtout(xtout[891]), .a(a[891]), .b(b[891]), .c(c[891]),
.sel(sel));
    xtmux3b i892(.xtout(xtout[892]), .a(a[892]), .b(b[892]), .c(c[892]),
.sel(sel));
    xtmux3b i893(.xtout(xtout[893]), .a(a[893]), .b(b[893]), .c(c[893]),
.sel(sel));
    xtmux3b i894(.xtout(xtout[894]), .a(a[894]), .b(b[894]), .c(c[894]),
.sel(sel));
    xtmux3b i895(.xtout(xtout[895]), .a(a[895]), .b(b[895]), .c(c[895]),
.sel(sel));
    xtmux3b i896(.xtout(xtout[896]), .a(a[896]), .b(b[896]), .c(c[896]),
.sel(sel));
    xtmux3b i897(.xtout(xtout[897]), .a(a[897]), .b(b[897]), .c(c[897]),
.sel(sel));
    xtmux3b i898(.xtout(xtout[898]), .a(a[898]), .b(b[898]), .c(c[898]),
.sel(sel));
    xtmux3b i899(.xtout(xtout[899]), .a(a[899]), .b(b[899]), .c(c[899]),
.sel(sel));
```

```
    xtmux3b i900(.xtout(xtout[900]), .a(a[900]), .b(b[900]), .c(c[900]),
.sel(sel));
    xtmux3b i901(.xtout(xtout[901]), .a(a[901]), .b(b[901]), .c(c[901]),
.sel(sel));
    xtmux3b i902(.xtout(xtout[902]), .a(a[902]), .b(b[902]), .c(c[902]),
.sel(sel));
    xtmux3b i903(.xtout(xtout[903]), .a(a[903]), .b(b[903]), .c(c[903]),
.sel(sel));
    xtmux3b i904(.xtout(xtout[904]), .a(a[904]), .b(b[904]), .c(c[904]),
.sel(sel));
    xtmux3b i905(.xtout(xtout[905]), .a(a[905]), .b(b[905]), .c(c[905]),
.sel(sel));
    xtmux3b i906(.xtout(xtout[906]), .a(a[906]), .b(b[906]), .c(c[906]),
.sel(sel));
    xtmux3b i907(.xtout(xtout[907]), .a(a[907]), .b(b[907]), .c(c[907]),
.sel(sel));
    xtmux3b i908(.xtout(xtout[908]), .a(a[908]), .b(b[908]), .c(c[908]),
.sel(sel));
    xtmux3b i909(.xtout(xtout[909]), .a(a[909]), .b(b[909]), .c(c[909]),
.sel(sel));
    xtmux3b i910(.xtout(xtout[910]), .a(a[910]), .b(b[910]), .c(c[910]),
.sel(sel));
    xtmux3b i911(.xtout(xtout[911]), .a(a[911]), .b(b[911]), .c(c[911]),
.sel(sel));
    xtmux3b i912(.xtout(xtout[912]), .a(a[912]), .b(b[912]), .c(c[912]),
.sel(sel));
    xtmux3b i913(.xtout(xtout[913]), .a(a[913]), .b(b[913]), .c(c[913]),
.sel(sel));
    xtmux3b i914(.xtout(xtout[914]), .a(a[914]), .b(b[914]), .c(c[914]),
.sel(sel));
    xtmux3b i915(.xtout(xtout[915]), .a(a[915]), .b(b[915]), .c(c[915]),
.sel(sel));
    xtmux3b i916(.xtout(xtout[916]), .a(a[916]), .b(b[916]), .c(c[916]),
.sel(sel));
    xtmux3b i917(.xtout(xtout[917]), .a(a[917]), .b(b[917]), .c(c[917]),
.sel(sel));
    xtmux3b i918(.xtout(xtout[918]), .a(a[918]), .b(b[918]), .c(c[918]),
.sel(sel));
    xtmux3b i919(.xtout(xtout[919]), .a(a[919]), .b(b[919]), .c(c[919]),
.sel(sel));
    xtmux3b i920(.xtout(xtout[920]), .a(a[920]), .b(b[920]), .c(c[920]),
.sel(sel));
    xtmux3b i921(.xtout(xtout[921]), .a(a[921]), .b(b[921]), .c(c[921]),
.sel(sel));
    xtmux3b i922(.xtout(xtout[922]), .a(a[922]), .b(b[922]), .c(c[922]),
.sel(sel));
    xtmux3b i923(.xtout(xtout[923]), .a(a[923]), .b(b[923]), .c(c[923]),
.sel(sel));
    xtmux3b i924(.xtout(xtout[924]), .a(a[924]), .b(b[924]), .c(c[924]),
.sel(sel));
    xtmux3b i925(.xtout(xtout[925]), .a(a[925]), .b(b[925]), .c(c[925]),
.sel(sel));
    xtmux3b i926(.xtout(xtout[926]), .a(a[926]), .b(b[926]), .c(c[926]),
.sel(sel));
    xtmux3b i927(.xtout(xtout[927]), .a(a[927]), .b(b[927]), .c(c[927]),
.sel(sel));
```

```
    xtmux3b i928(.xtout(xtout[928]), .a(a[928]), .b(b[928]), .c(c[928]),
.sel(sel));
    xtmux3b i929(.xtout(xtout[929]), .a(a[929]), .b(b[929]), .c(c[929]),
.sel(sel));
    xtmux3b i930(.xtout(xtout[930]), .a(a[930]), .b(b[930]), .c(c[930]),
.sel(sel));
    xtmux3b i931(.xtout(xtout[931]), .a(a[931]), .b(b[931]), .c(c[931]),
.sel(sel));
    xtmux3b i932(.xtout(xtout[932]), .a(a[932]), .b(b[932]), .c(c[932]),
.sel(sel));
    xtmux3b i933(.xtout(xtout[933]), .a(a[933]), .b(b[933]), .c(c[933]),
.sel(sel));
    xtmux3b i934(.xtout(xtout[934]), .a(a[934]), .b(b[934]), .c(c[934]),
.sel(sel));
    xtmux3b i935(.xtout(xtout[935]), .a(a[935]), .b(b[935]), .c(c[935]),
.sel(sel));
    xtmux3b i936(.xtout(xtout[936]), .a(a[936]), .b(b[936]), .c(c[936]),
.sel(sel));
    xtmux3b i937(.xtout(xtout[937]), .a(a[937]), .b(b[937]), .c(c[937]),
.sel(sel));
    xtmux3b i938(.xtout(xtout[938]), .a(a[938]), .b(b[938]), .c(c[938]),
.sel(sel));
    xtmux3b i939(.xtout(xtout[939]), .a(a[939]), .b(b[939]), .c(c[939]),
.sel(sel));
    xtmux3b i940(.xtout(xtout[940]), .a(a[940]), .b(b[940]), .c(c[940]),
.sel(sel));
    xtmux3b i941(.xtout(xtout[941]), .a(a[941]), .b(b[941]), .c(c[941]),
.sel(sel));
    xtmux3b i942(.xtout(xtout[942]), .a(a[942]), .b(b[942]), .c(c[942]),
.sel(sel));
    xtmux3b i943(.xtout(xtout[943]), .a(a[943]), .b(b[943]), .c(c[943]),
.sel(sel));
    xtmux3b i944(.xtout(xtout[944]), .a(a[944]), .b(b[944]), .c(c[944]),
.sel(sel));
    xtmux3b i945(.xtout(xtout[945]), .a(a[945]), .b(b[945]), .c(c[945]),
.sel(sel));
    xtmux3b i946(.xtout(xtout[946]), .a(a[946]), .b(b[946]), .c(c[946]),
.sel(sel));
    xtmux3b i947(.xtout(xtout[947]), .a(a[947]), .b(b[947]), .c(c[947]),
.sel(sel));
    xtmux3b i948(.xtout(xtout[948]), .a(a[948]), .b(b[948]), .c(c[948]),
.sel(sel));
    xtmux3b i949(.xtout(xtout[949]), .a(a[949]), .b(b[949]), .c(c[949]),
.sel(sel));
    xtmux3b i950(.xtout(xtout[950]), .a(a[950]), .b(b[950]), .c(c[950]),
.sel(sel));
    xtmux3b i951(.xtout(xtout[951]), .a(a[951]), .b(b[951]), .c(c[951]),
.sel(sel));
    xtmux3b i952(.xtout(xtout[952]), .a(a[952]), .b(b[952]), .c(c[952]),
.sel(sel));
    xtmux3b i953(.xtout(xtout[953]), .a(a[953]), .b(b[953]), .c(c[953]),
.sel(sel));
    xtmux3b i954(.xtout(xtout[954]), .a(a[954]), .b(b[954]), .c(c[954]),
.sel(sel));
    xtmux3b i955(.xtout(xtout[955]), .a(a[955]), .b(b[955]), .c(c[955]),
.sel(sel));
```

```
    xtmux3b i956(.xtout(xtout[956]), .a(a[956]), .b(b[956]), .c(c[956]),
.sel(sel));
    xtmux3b i957(.xtout(xtout[957]), .a(a[957]), .b(b[957]), .c(c[957]),
.sel(sel));
    xtmux3b i958(.xtout(xtout[958]), .a(a[958]), .b(b[958]), .c(c[958]),
.sel(sel));
    xtmux3b i959(.xtout(xtout[959]), .a(a[959]), .b(b[959]), .c(c[959]),
.sel(sel));
    xtmux3b i960(.xtout(xtout[960]), .a(a[960]), .b(b[960]), .c(c[960]),
.sel(sel));
    xtmux3b i961(.xtout(xtout[961]), .a(a[961]), .b(b[961]), .c(c[961]),
.sel(sel));
    xtmux3b i962(.xtout(xtout[962]), .a(a[962]), .b(b[962]), .c(c[962]),
.sel(sel));
    xtmux3b i963(.xtout(xtout[963]), .a(a[963]), .b(b[963]), .c(c[963]),
.sel(sel));
    xtmux3b i964(.xtout(xtout[964]), .a(a[964]), .b(b[964]), .c(c[964]),
.sel(sel));
    xtmux3b i965(.xtout(xtout[965]), .a(a[965]), .b(b[965]), .c(c[965]),
.sel(sel));
    xtmux3b i966(.xtout(xtout[966]), .a(a[966]), .b(b[966]), .c(c[966]),
.sel(sel));
    xtmux3b i967(.xtout(xtout[967]), .a(a[967]), .b(b[967]), .c(c[967]),
.sel(sel));
    xtmux3b i968(.xtout(xtout[968]), .a(a[968]), .b(b[968]), .c(c[968]),
.sel(sel));
    xtmux3b i969(.xtout(xtout[969]), .a(a[969]), .b(b[969]), .c(c[969]),
.sel(sel));
    xtmux3b i970(.xtout(xtout[970]), .a(a[970]), .b(b[970]), .c(c[970]),
.sel(sel));
    xtmux3b i971(.xtout(xtout[971]), .a(a[971]), .b(b[971]), .c(c[971]),
.sel(sel));
    xtmux3b i972(.xtout(xtout[972]), .a(a[972]), .b(b[972]), .c(c[972]),
.sel(sel));
    xtmux3b i973(.xtout(xtout[973]), .a(a[973]), .b(b[973]), .c(c[973]),
.sel(sel));
    xtmux3b i974(.xtout(xtout[974]), .a(a[974]), .b(b[974]), .c(c[974]),
.sel(sel));
    xtmux3b i975(.xtout(xtout[975]), .a(a[975]), .b(b[975]), .c(c[975]),
.sel(sel));
    xtmux3b i976(.xtout(xtout[976]), .a(a[976]), .b(b[976]), .c(c[976]),
.sel(sel));
    xtmux3b i977(.xtout(xtout[977]), .a(a[977]), .b(b[977]), .c(c[977]),
.sel(sel));
    xtmux3b i978(.xtout(xtout[978]), .a(a[978]), .b(b[978]), .c(c[978]),
.sel(sel));
    xtmux3b i979(.xtout(xtout[979]), .a(a[979]), .b(b[979]), .c(c[979]),
.sel(sel));
    xtmux3b i980(.xtout(xtout[980]), .a(a[980]), .b(b[980]), .c(c[980]),
.sel(sel));
    xtmux3b i981(.xtout(xtout[981]), .a(a[981]), .b(b[981]), .c(c[981]),
.sel(sel));
    xtmux3b i982(.xtout(xtout[982]), .a(a[982]), .b(b[982]), .c(c[982]),
.sel(sel));
    xtmux3b i983(.xtout(xtout[983]), .a(a[983]), .b(b[983]), .c(c[983]),
.sel(sel));
```

```
    xtmux3b i984(.xtout(xtout[984]), .a(a[984]), .b(b[984]), .c(c[984]),
.sel(sel));
    xtmux3b i985(.xtout(xtout[985]), .a(a[985]), .b(b[985]), .c(c[985]),
.sel(sel));
    xtmux3b i986(.xtout(xtout[986]), .a(a[986]), .b(b[986]), .c(c[986]),
.sel(sel));
    xtmux3b i987(.xtout(xtout[987]), .a(a[987]), .b(b[987]), .c(c[987]),
.sel(sel));
    xtmux3b i988(.xtout(xtout[988]), .a(a[988]), .b(b[988]), .c(c[988]),
.sel(sel));
    xtmux3b i989(.xtout(xtout[989]), .a(a[989]), .b(b[989]), .c(c[989]),
.sel(sel));
    xtmux3b i990(.xtout(xtout[990]), .a(a[990]), .b(b[990]), .c(c[990]),
.sel(sel));
    xtmux3b i991(.xtout(xtout[991]), .a(a[991]), .b(b[991]), .c(c[991]),
.sel(sel));
    xtmux3b i992(.xtout(xtout[992]), .a(a[992]), .b(b[992]), .c(c[992]),
.sel(sel));
    xtmux3b i993(.xtout(xtout[993]), .a(a[993]), .b(b[993]), .c(c[993]),
.sel(sel));
    xtmux3b i994(.xtout(xtout[994]), .a(a[994]), .b(b[994]), .c(c[994]),
.sel(sel));
    xtmux3b i995(.xtout(xtout[995]), .a(a[995]), .b(b[995]), .c(c[995]),
.sel(sel));
    xtmux3b i996(.xtout(xtout[996]), .a(a[996]), .b(b[996]), .c(c[996]),
.sel(sel));
    xtmux3b i997(.xtout(xtout[997]), .a(a[997]), .b(b[997]), .c(c[997]),
.sel(sel));
    xtmux3b i998(.xtout(xtout[998]), .a(a[998]), .b(b[998]), .c(c[998]),
.sel(sel));
    xtmux3b i999(.xtout(xtout[999]), .a(a[999]), .b(b[999]), .c(c[999]),
.sel(sel));
    xtmux3b i1000(.xtout(xtout[1000]), .a(a[1000]), .b(b[1000]), .c(c[1000]),
.sel(sel));
    xtmux3b i1001(.xtout(xtout[1001]), .a(a[1001]), .b(b[1001]), .c(c[1001]),
.sel(sel));
    xtmux3b i1002(.xtout(xtout[1002]), .a(a[1002]), .b(b[1002]), .c(c[1002]),
.sel(sel));
    xtmux3b i1003(.xtout(xtout[1003]), .a(a[1003]), .b(b[1003]), .c(c[1003]),
.sel(sel));
    xtmux3b i1004(.xtout(xtout[1004]), .a(a[1004]), .b(b[1004]), .c(c[1004]),
.sel(sel));
    xtmux3b i1005(.xtout(xtout[1005]), .a(a[1005]), .b(b[1005]), .c(c[1005]),
.sel(sel));
    xtmux3b i1006(.xtout(xtout[1006]), .a(a[1006]), .b(b[1006]), .c(c[1006]),
.sel(sel));
    xtmux3b i1007(.xtout(xtout[1007]), .a(a[1007]), .b(b[1007]), .c(c[1007]),
.sel(sel));
    xtmux3b i1008(.xtout(xtout[1008]), .a(a[1008]), .b(b[1008]), .c(c[1008]),
.sel(sel));
    xtmux3b i1009(.xtout(xtout[1009]), .a(a[1009]), .b(b[1009]), .c(c[1009]),
.sel(sel));
    xtmux3b i1010(.xtout(xtout[1010]), .a(a[1010]), .b(b[1010]), .c(c[1010]),
.sel(sel));
    xtmux3b i1011(.xtout(xtout[1011]), .a(a[1011]), .b(b[1011]), .c(c[1011]),
.sel(sel));
```

```
    xtmux3b i1012(.xtout(xtout[1012]), .a(a[1012]), .b(b[1012]), .c(c[1012]),
.sel(sel));
    xtmux3b i1013(.xtout(xtout[1013]), .a(a[1013]), .b(b[1013]), .c(c[1013]),
.sel(sel));
    xtmux3b i1014(.xtout(xtout[1014]), .a(a[1014]), .b(b[1014]), .c(c[1014]),
.sel(sel));
    xtmux3b i1015(.xtout(xtout[1015]), .a(a[1015]), .b(b[1015]), .c(c[1015]),
.sel(sel));
    xtmux3b i1016(.xtout(xtout[1016]), .a(a[1016]), .b(b[1016]), .c(c[1016]),
.sel(sel));
    xtmux3b i1017(.xtout(xtout[1017]), .a(a[1017]), .b(b[1017]), .c(c[1017]),
.sel(sel));
    xtmux3b i1018(.xtout(xtout[1018]), .a(a[1018]), .b(b[1018]), .c(c[1018]),
.sel(sel));
    xtmux3b i1019(.xtout(xtout[1019]), .a(a[1019]), .b(b[1019]), .c(c[1019]),
.sel(sel));
    xtmux3b i1020(.xtout(xtout[1020]), .a(a[1020]), .b(b[1020]), .c(c[1020]),
.sel(sel));
    xtmux3b i1021(.xtout(xtout[1021]), .a(a[1021]), .b(b[1021]), .c(c[1021]),
.sel(sel));
    xtmux3b i1022(.xtout(xtout[1022]), .a(a[1022]), .b(b[1022]), .c(c[1022]),
.sel(sel));
    xtmux3b i1023(.xtout(xtout[1023]), .a(a[1023]), .b(b[1023]), .c(c[1023]),
.sel(sel));
endmodule
module xtmux4e_1024(xtout, a, b, c, d, sel);
output [1023:0] xtout;
input [1023:0] a, b, c, d;
input [1:0] sel;
    xtmux4b i0(.xtout(xtout[0]), .a(a[0]), .b(b[0]), .c(c[0]), .d(d[0]),
.sel(sel));
    xtmux4b i1(.xtout(xtout[1]), .a(a[1]), .b(b[1]), .c(c[1]), .d(d[1]),
.sel(sel));
    xtmux4b i2(.xtout(xtout[2]), .a(a[2]), .b(b[2]), .c(c[2]), .d(d[2]),
.sel(sel));
    xtmux4b i3(.xtout(xtout[3]), .a(a[3]), .b(b[3]), .c(c[3]), .d(d[3]),
.sel(sel));
    xtmux4b i4(.xtout(xtout[4]), .a(a[4]), .b(b[4]), .c(c[4]), .d(d[4]),
.sel(sel));
    xtmux4b i5(.xtout(xtout[5]), .a(a[5]), .b(b[5]), .c(c[5]), .d(d[5]),
.sel(sel));
    xtmux4b i6(.xtout(xtout[6]), .a(a[6]), .b(b[6]), .c(c[6]), .d(d[6]),
.sel(sel));
    xtmux4b i7(.xtout(xtout[7]), .a(a[7]), .b(b[7]), .c(c[7]), .d(d[7]),
.sel(sel));
    xtmux4b i8(.xtout(xtout[8]), .a(a[8]), .b(b[8]), .c(c[8]), .d(d[8]),
.sel(sel));
    xtmux4b i9(.xtout(xtout[9]), .a(a[9]), .b(b[9]), .c(c[9]), .d(d[9]),
.sel(sel));
    xtmux4b i10(.xtout(xtout[10]), .a(a[10]), .b(b[10]), .c(c[10]), .d(d[10]),
.sel(sel));
    xtmux4b i11(.xtout(xtout[11]), .a(a[11]), .b(b[11]), .c(c[11]), .d(d[11]),
.sel(sel));
    xtmux4b i12(.xtout(xtout[12]), .a(a[12]), .b(b[12]), .c(c[12]), .d(d[12]),
.sel(sel));
    xtmux4b i13(.xtout(xtout[13]), .a(a[13]), .b(b[13]), .c(c[13]), .d(d[13]),
.sel(sel));
```

```
    xtmux4b i14(.xtout(xtout[14]), .a(a[14]), .b(b[14]), .c(c[14]), .d(d[14]),
.sel(sel));
    xtmux4b i15(.xtout(xtout[15]), .a(a[15]), .b(b[15]), .c(c[15]), .d(d[15]),
.sel(sel));
    xtmux4b i16(.xtout(xtout[16]), .a(a[16]), .b(b[16]), .c(c[16]), .d(d[16]),
.sel(sel));
    xtmux4b i17(.xtout(xtout[17]), .a(a[17]), .b(b[17]), .c(c[17]), .d(d[17]),
.sel(sel));
    xtmux4b i18(.xtout(xtout[18]), .a(a[18]), .b(b[18]), .c(c[18]), .d(d[18]),
.sel(sel));
    xtmux4b i19(.xtout(xtout[19]), .a(a[19]), .b(b[19]), .c(c[19]), .d(d[19]),
.sel(sel));
    xtmux4b i20(.xtout(xtout[20]), .a(a[20]), .b(b[20]), .c(c[20]), .d(d[20]),
.sel(sel));
    xtmux4b i21(.xtout(xtout[21]), .a(a[21]), .b(b[21]), .c(c[21]), .d(d[21]),
.sel(sel));
    xtmux4b i22(.xtout(xtout[22]), .a(a[22]), .b(b[22]), .c(c[22]), .d(d[22]),
.sel(sel));
    xtmux4b i23(.xtout(xtout[23]), .a(a[23]), .b(b[23]), .c(c[23]), .d(d[23]),
.sel(sel));
    xtmux4b i24(.xtout(xtout[24]), .a(a[24]), .b(b[24]), .c(c[24]), .d(d[24]),
.sel(sel));
    xtmux4b i25(.xtout(xtout[25]), .a(a[25]), .b(b[25]), .c(c[25]), .d(d[25]),
.sel(sel));
    xtmux4b i26(.xtout(xtout[26]), .a(a[26]), .b(b[26]), .c(c[26]), .d(d[26]),
.sel(sel));
    xtmux4b i27(.xtout(xtout[27]), .a(a[27]), .b(b[27]), .c(c[27]), .d(d[27]),
.sel(sel));
    xtmux4b i28(.xtout(xtout[28]), .a(a[28]), .b(b[28]), .c(c[28]), .d(d[28]),
.sel(sel));
    xtmux4b i29(.xtout(xtout[29]), .a(a[29]), .b(b[29]), .c(c[29]), .d(d[29]),
.sel(sel));
    xtmux4b i30(.xtout(xtout[30]), .a(a[30]), .b(b[30]), .c(c[30]), .d(d[30]),
.sel(sel));
    xtmux4b i31(.xtout(xtout[31]), .a(a[31]), .b(b[31]), .c(c[31]), .d(d[31]),
.sel(sel));
    xtmux4b i32(.xtout(xtout[32]), .a(a[32]), .b(b[32]), .c(c[32]), .d(d[32]),
.sel(sel));
    xtmux4b i33(.xtout(xtout[33]), .a(a[33]), .b(b[33]), .c(c[33]), .d(d[33]),
.sel(sel));
    xtmux4b i34(.xtout(xtout[34]), .a(a[34]), .b(b[34]), .c(c[34]), .d(d[34]),
.sel(sel));
    xtmux4b i35(.xtout(xtout[35]), .a(a[35]), .b(b[35]), .c(c[35]), .d(d[35]),
.sel(sel));
    xtmux4b i36(.xtout(xtout[36]), .a(a[36]), .b(b[36]), .c(c[36]), .d(d[36]),
.sel(sel));
    xtmux4b i37(.xtout(xtout[37]), .a(a[37]), .b(b[37]), .c(c[37]), .d(d[37]),
.sel(sel));
    xtmux4b i38(.xtout(xtout[38]), .a(a[38]), .b(b[38]), .c(c[38]), .d(d[38]),
.sel(sel));
    xtmux4b i39(.xtout(xtout[39]), .a(a[39]), .b(b[39]), .c(c[39]), .d(d[39]),
.sel(sel));
    xtmux4b i40(.xtout(xtout[40]), .a(a[40]), .b(b[40]), .c(c[40]), .d(d[40]),
.sel(sel));
    xtmux4b i41(.xtout(xtout[41]), .a(a[41]), .b(b[41]), .c(c[41]), .d(d[41]),
.sel(sel));
```

```
    xtmux4b i42(.xtout(xtout[42]), .a(a[42]), .b(b[42]), .c(c[42]), .d(d[42]),
.sel(sel));
    xtmux4b i43(.xtout(xtout[43]), .a(a[43]), .b(b[43]), .c(c[43]), .d(d[43]),
.sel(sel));
    xtmux4b i44(.xtout(xtout[44]), .a(a[44]), .b(b[44]), .c(c[44]), .d(d[44]),
.sel(sel));
    xtmux4b i45(.xtout(xtout[45]), .a(a[45]), .b(b[45]), .c(c[45]), .d(d[45]),
.sel(sel));
    xtmux4b i46(.xtout(xtout[46]), .a(a[46]), .b(b[46]), .c(c[46]), .d(d[46]),
.sel(sel));
    xtmux4b i47(.xtout(xtout[47]), .a(a[47]), .b(b[47]), .c(c[47]), .d(d[47]),
.sel(sel));
    xtmux4b i48(.xtout(xtout[48]), .a(a[48]), .b(b[48]), .c(c[48]), .d(d[48]),
.sel(sel));
    xtmux4b i49(.xtout(xtout[49]), .a(a[49]), .b(b[49]), .c(c[49]), .d(d[49]),
.sel(sel));
    xtmux4b i50(.xtout(xtout[50]), .a(a[50]), .b(b[50]), .c(c[50]), .d(d[50]),
.sel(sel));
    xtmux4b i51(.xtout(xtout[51]), .a(a[51]), .b(b[51]), .c(c[51]), .d(d[51]),
.sel(sel));
    xtmux4b i52(.xtout(xtout[52]), .a(a[52]), .b(b[52]), .c(c[52]), .d(d[52]),
.sel(sel));
    xtmux4b i53(.xtout(xtout[53]), .a(a[53]), .b(b[53]), .c(c[53]), .d(d[53]),
.sel(sel));
    xtmux4b i54(.xtout(xtout[54]), .a(a[54]), .b(b[54]), .c(c[54]), .d(d[54]),
.sel(sel));
    xtmux4b i55(.xtout(xtout[55]), .a(a[55]), .b(b[55]), .c(c[55]), .d(d[55]),
.sel(sel));
    xtmux4b i56(.xtout(xtout[56]), .a(a[56]), .b(b[56]), .c(c[56]), .d(d[56]),
.sel(sel));
    xtmux4b i57(.xtout(xtout[57]), .a(a[57]), .b(b[57]), .c(c[57]), .d(d[57]),
.sel(sel));
    xtmux4b i58(.xtout(xtout[58]), .a(a[58]), .b(b[58]), .c(c[58]), .d(d[58]),
.sel(sel));
    xtmux4b i59(.xtout(xtout[59]), .a(a[59]), .b(b[59]), .c(c[59]), .d(d[59]),
.sel(sel));
    xtmux4b i60(.xtout(xtout[60]), .a(a[60]), .b(b[60]), .c(c[60]), .d(d[60]),
.sel(sel));
    xtmux4b i61(.xtout(xtout[61]), .a(a[61]), .b(b[61]), .c(c[61]), .d(d[61]),
.sel(sel));
    xtmux4b i62(.xtout(xtout[62]), .a(a[62]), .b(b[62]), .c(c[62]), .d(d[62]),
.sel(sel));
    xtmux4b i63(.xtout(xtout[63]), .a(a[63]), .b(b[63]), .c(c[63]), .d(d[63]),
.sel(sel));
    xtmux4b i64(.xtout(xtout[64]), .a(a[64]), .b(b[64]), .c(c[64]), .d(d[64]),
.sel(sel));
    xtmux4b i65(.xtout(xtout[65]), .a(a[65]), .b(b[65]), .c(c[65]), .d(d[65]),
.sel(sel));
    xtmux4b i66(.xtout(xtout[66]), .a(a[66]), .b(b[66]), .c(c[66]), .d(d[66]),
.sel(sel));
    xtmux4b i67(.xtout(xtout[67]), .a(a[67]), .b(b[67]), .c(c[67]), .d(d[67]),
.sel(sel));
    xtmux4b i68(.xtout(xtout[68]), .a(a[68]), .b(b[68]), .c(c[68]), .d(d[68]),
.sel(sel));
    xtmux4b i69(.xtout(xtout[69]), .a(a[69]), .b(b[69]), .c(c[69]), .d(d[69]),
.sel(sel));
```

```
    xtmux4b i70(.xtout(xtout[70]), .a(a[70]), .b(b[70]), .c(c[70]), .d(d[70]),
.sel(sel));
    xtmux4b i71(.xtout(xtout[71]), .a(a[71]), .b(b[71]), .c(c[71]), .d(d[71]),
.sel(sel));
    xtmux4b i72(.xtout(xtout[72]), .a(a[72]), .b(b[72]), .c(c[72]), .d(d[72]),
.sel(sel));
    xtmux4b i73(.xtout(xtout[73]), .a(a[73]), .b(b[73]), .c(c[73]), .d(d[73]),
.sel(sel));
    xtmux4b i74(.xtout(xtout[74]), .a(a[74]), .b(b[74]), .c(c[74]), .d(d[74]),
.sel(sel));
    xtmux4b i75(.xtout(xtout[75]), .a(a[75]), .b(b[75]), .c(c[75]), .d(d[75]),
.sel(sel));
    xtmux4b i76(.xtout(xtout[76]), .a(a[76]), .b(b[76]), .c(c[76]), .d(d[76]),
.sel(sel));
    xtmux4b i77(.xtout(xtout[77]), .a(a[77]), .b(b[77]), .c(c[77]), .d(d[77]),
.sel(sel));
    xtmux4b i78(.xtout(xtout[78]), .a(a[78]), .b(b[78]), .c(c[78]), .d(d[78]),
.sel(sel));
    xtmux4b i79(.xtout(xtout[79]), .a(a[79]), .b(b[79]), .c(c[79]), .d(d[79]),
.sel(sel));
    xtmux4b i80(.xtout(xtout[80]), .a(a[80]), .b(b[80]), .c(c[80]), .d(d[80]),
.sel(sel));
    xtmux4b i81(.xtout(xtout[81]), .a(a[81]), .b(b[81]), .c(c[81]), .d(d[81]),
.sel(sel));
    xtmux4b i82(.xtout(xtout[82]), .a(a[82]), .b(b[82]), .c(c[82]), .d(d[82]),
.sel(sel));
    xtmux4b i83(.xtout(xtout[83]), .a(a[83]), .b(b[83]), .c(c[83]), .d(d[83]),
.sel(sel));
    xtmux4b i84(.xtout(xtout[84]), .a(a[84]), .b(b[84]), .c(c[84]), .d(d[84]),
.sel(sel));
    xtmux4b i85(.xtout(xtout[85]), .a(a[85]), .b(b[85]), .c(c[85]), .d(d[85]),
.sel(sel));
    xtmux4b i86(.xtout(xtout[86]), .a(a[86]), .b(b[86]), .c(c[86]), .d(d[86]),
.sel(sel));
    xtmux4b i87(.xtout(xtout[87]), .a(a[87]), .b(b[87]), .c(c[87]), .d(d[87]),
.sel(sel));
    xtmux4b i88(.xtout(xtout[88]), .a(a[88]), .b(b[88]), .c(c[88]), .d(d[88]),
.sel(sel));
    xtmux4b i89(.xtout(xtout[89]), .a(a[89]), .b(b[89]), .c(c[89]), .d(d[89]),
.sel(sel));
    xtmux4b i90(.xtout(xtout[90]), .a(a[90]), .b(b[90]), .c(c[90]), .d(d[90]),
.sel(sel));
    xtmux4b i91(.xtout(xtout[91]), .a(a[91]), .b(b[91]), .c(c[91]), .d(d[91]),
.sel(sel));
    xtmux4b i92(.xtout(xtout[92]), .a(a[92]), .b(b[92]), .c(c[92]), .d(d[92]),
.sel(sel));
    xtmux4b i93(.xtout(xtout[93]), .a(a[93]), .b(b[93]), .c(c[93]), .d(d[93]),
.sel(sel));
    xtmux4b i94(.xtout(xtout[94]), .a(a[94]), .b(b[94]), .c(c[94]), .d(d[94]),
.sel(sel));
    xtmux4b i95(.xtout(xtout[95]), .a(a[95]), .b(b[95]), .c(c[95]), .d(d[95]),
.sel(sel));
    xtmux4b i96(.xtout(xtout[96]), .a(a[96]), .b(b[96]), .c(c[96]), .d(d[96]),
.sel(sel));
    xtmux4b i97(.xtout(xtout[97]), .a(a[97]), .b(b[97]), .c(c[97]), .d(d[97]),
.sel(sel));
```

```
    xtmux4b i98(.xtout(xtout[98]), .a(a[98]), .b(b[98]), .c(c[98]), .d(d[98]),
.sel(sel));
    xtmux4b i99(.xtout(xtout[99]), .a(a[99]), .b(b[99]), .c(c[99]), .d(d[99]),
.sel(sel));
    xtmux4b i100(.xtout(xtout[100]), .a(a[100]), .b(b[100]), .c(c[100]),
.d(d[100]), .sel(sel));
    xtmux4b i101(.xtout(xtout[101]), .a(a[101]), .b(b[101]), .c(c[101]),
.d(d[101]), .sel(sel));
    xtmux4b i102(.xtout(xtout[102]), .a(a[102]), .b(b[102]), .c(c[102]),
.d(d[102]), .sel(sel));
    xtmux4b i103(.xtout(xtout[103]), .a(a[103]), .b(b[103]), .c(c[103]),
.d(d[103]), .sel(sel));
    xtmux4b i104(.xtout(xtout[104]), .a(a[104]), .b(b[104]), .c(c[104]),
.d(d[104]), .sel(sel));
    xtmux4b i105(.xtout(xtout[105]), .a(a[105]), .b(b[105]), .c(c[105]),
.d(d[105]), .sel(sel));
    xtmux4b i106(.xtout(xtout[106]), .a(a[106]), .b(b[106]), .c(c[106]),
.d(d[106]), .sel(sel));
    xtmux4b i107(.xtout(xtout[107]), .a(a[107]), .b(b[107]), .c(c[107]),
.d(d[107]), .sel(sel));
    xtmux4b i108(.xtout(xtout[108]), .a(a[108]), .b(b[108]), .c(c[108]),
.d(d[108]), .sel(sel));
    xtmux4b i109(.xtout(xtout[109]), .a(a[109]), .b(b[109]), .c(c[109]),
.d(d[109]), .sel(sel));
    xtmux4b i110(.xtout(xtout[110]), .a(a[110]), .b(b[110]), .c(c[110]),
.d(d[110]), .sel(sel));
    xtmux4b i111(.xtout(xtout[111]), .a(a[111]), .b(b[111]), .c(c[111]),
.d(d[111]), .sel(sel));
    xtmux4b i112(.xtout(xtout[112]), .a(a[112]), .b(b[112]), .c(c[112]),
.d(d[112]), .sel(sel));
    xtmux4b i113(.xtout(xtout[113]), .a(a[113]), .b(b[113]), .c(c[113]),
.d(d[113]), .sel(sel));
    xtmux4b i114(.xtout(xtout[114]), .a(a[114]), .b(b[114]), .c(c[114]),
.d(d[114]), .sel(sel));
    xtmux4b i115(.xtout(xtout[115]), .a(a[115]), .b(b[115]), .c(c[115]),
.d(d[115]), .sel(sel));
    xtmux4b i116(.xtout(xtout[116]), .a(a[116]), .b(b[116]), .c(c[116]),
.d(d[116]), .sel(sel));
    xtmux4b i117(.xtout(xtout[117]), .a(a[117]), .b(b[117]), .c(c[117]),
.d(d[117]), .sel(sel));
    xtmux4b i118(.xtout(xtout[118]), .a(a[118]), .b(b[118]), .c(c[118]),
.d(d[118]), .sel(sel));
    xtmux4b i119(.xtout(xtout[119]), .a(a[119]), .b(b[119]), .c(c[119]),
.d(d[119]), .sel(sel));
    xtmux4b i120(.xtout(xtout[120]), .a(a[120]), .b(b[120]), .c(c[120]),
.d(d[120]), .sel(sel));
    xtmux4b i121(.xtout(xtout[121]), .a(a[121]), .b(b[121]), .c(c[121]),
.d(d[121]), .sel(sel));
    xtmux4b i122(.xtout(xtout[122]), .a(a[122]), .b(b[122]), .c(c[122]),
.d(d[122]), .sel(sel));
    xtmux4b i123(.xtout(xtout[123]), .a(a[123]), .b(b[123]), .c(c[123]),
.d(d[123]), .sel(sel));
    xtmux4b i124(.xtout(xtout[124]), .a(a[124]), .b(b[124]), .c(c[124]),
.d(d[124]), .sel(sel));
    xtmux4b i125(.xtout(xtout[125]), .a(a[125]), .b(b[125]), .c(c[125]),
.d(d[125]), .sel(sel));
```

```
    xtmux4b i126(.xtout(xtout[126]), .a(a[126]), .b(b[126]), .c(c[126]),
.d(d[126]), .sel(sel));
    xtmux4b i127(.xtout(xtout[127]), .a(a[127]), .b(b[127]), .c(c[127]),
.d(d[127]), .sel(sel));
    xtmux4b i128(.xtout(xtout[128]), .a(a[128]), .b(b[128]), .c(c[128]),
.d(d[128]), .sel(sel));
    xtmux4b i129(.xtout(xtout[129]), .a(a[129]), .b(b[129]), .c(c[129]),
.d(d[129]), .sel(sel));
    xtmux4b i130(.xtout(xtout[130]), .a(a[130]), .b(b[130]), .c(c[130]),
.d(d[130]), .sel(sel));
    xtmux4b i131(.xtout(xtout[131]), .a(a[131]), .b(b[131]), .c(c[131]),
.d(d[131]), .sel(sel));
    xtmux4b i132(.xtout(xtout[132]), .a(a[132]), .b(b[132]), .c(c[132]),
.d(d[132]), .sel(sel));
    xtmux4b i133(.xtout(xtout[133]), .a(a[133]), .b(b[133]), .c(c[133]),
.d(d[133]), .sel(sel));
    xtmux4b i134(.xtout(xtout[134]), .a(a[134]), .b(b[134]), .c(c[134]),
.d(d[134]), .sel(sel));
    xtmux4b i135(.xtout(xtout[135]), .a(a[135]), .b(b[135]), .c(c[135]),
.d(d[135]), .sel(sel));
    xtmux4b i136(.xtout(xtout[136]), .a(a[136]), .b(b[136]), .c(c[136]),
.d(d[136]), .sel(sel));
    xtmux4b i137(.xtout(xtout[137]), .a(a[137]), .b(b[137]), .c(c[137]),
.d(d[137]), .sel(sel));
    xtmux4b i138(.xtout(xtout[138]), .a(a[138]), .b(b[138]), .c(c[138]),
.d(d[138]), .sel(sel));
    xtmux4b i139(.xtout(xtout[139]), .a(a[139]), .b(b[139]), .c(c[139]),
.d(d[139]), .sel(sel));
    xtmux4b i140(.xtout(xtout[140]), .a(a[140]), .b(b[140]), .c(c[140]),
.d(d[140]), .sel(sel));
    xtmux4b i141(.xtout(xtout[141]), .a(a[141]), .b(b[141]), .c(c[141]),
.d(d[141]), .sel(sel));
    xtmux4b i142(.xtout(xtout[142]), .a(a[142]), .b(b[142]), .c(c[142]),
.d(d[142]), .sel(sel));
    xtmux4b i143(.xtout(xtout[143]), .a(a[143]), .b(b[143]), .c(c[143]),
.d(d[143]), .sel(sel));
    xtmux4b i144(.xtout(xtout[144]), .a(a[144]), .b(b[144]), .c(c[144]),
.d(d[144]), .sel(sel));
    xtmux4b i145(.xtout(xtout[145]), .a(a[145]), .b(b[145]), .c(c[145]),
.d(d[145]), .sel(sel));
    xtmux4b i146(.xtout(xtout[146]), .a(a[146]), .b(b[146]), .c(c[146]),
.d(d[146]), .sel(sel));
    xtmux4b i147(.xtout(xtout[147]), .a(a[147]), .b(b[147]), .c(c[147]),
.d(d[147]), .sel(sel));
    xtmux4b i148(.xtout(xtout[148]), .a(a[148]), .b(b[148]), .c(c[148]),
.d(d[148]), .sel(sel));
    xtmux4b i149(.xtout(xtout[149]), .a(a[149]), .b(b[149]), .c(c[149]),
.d(d[149]), .sel(sel));
    xtmux4b i150(.xtout(xtout[150]), .a(a[150]), .b(b[150]), .c(c[150]),
.d(d[150]), .sel(sel));
    xtmux4b i151(.xtout(xtout[151]), .a(a[151]), .b(b[151]), .c(c[151]),
.d(d[151]), .sel(sel));
    xtmux4b i152(.xtout(xtout[152]), .a(a[152]), .b(b[152]), .c(c[152]),
.d(d[152]), .sel(sel));
    xtmux4b i153(.xtout(xtout[153]), .a(a[153]), .b(b[153]), .c(c[153]),
.d(d[153]), .sel(sel));
```

```
        xtmux4b i154(.xtout(xtout[154]), .a(a[154]), .b(b[154]), .c(c[154]),
.d(d[154]), .sel(sel));
        xtmux4b i155(.xtout(xtout[155]), .a(a[155]), .b(b[155]), .c(c[155]),
.d(d[155]), .sel(sel));
        xtmux4b i156(.xtout(xtout[156]), .a(a[156]), .b(b[156]), .c(c[156]),
.d(d[156]), .sel(sel));
        xtmux4b i157(.xtout(xtout[157]), .a(a[157]), .b(b[157]), .c(c[157]),
.d(d[157]), .sel(sel));
        xtmux4b i158(.xtout(xtout[158]), .a(a[158]), .b(b[158]), .c(c[158]),
.d(d[158]), .sel(sel));
        xtmux4b i159(.xtout(xtout[159]), .a(a[159]), .b(b[159]), .c(c[159]),
.d(d[159]), .sel(sel));
        xtmux4b i160(.xtout(xtout[160]), .a(a[160]), .b(b[160]), .c(c[160]),
.d(d[160]), .sel(sel));
        xtmux4b i161(.xtout(xtout[161]), .a(a[161]), .b(b[161]), .c(c[161]),
.d(d[161]), .sel(sel));
        xtmux4b i162(.xtout(xtout[162]), .a(a[162]), .b(b[162]), .c(c[162]),
.d(d[162]), .sel(sel));
        xtmux4b i163(.xtout(xtout[163]), .a(a[163]), .b(b[163]), .c(c[163]),
.d(d[163]), .sel(sel));
        xtmux4b i164(.xtout(xtout[164]), .a(a[164]), .b(b[164]), .c(c[164]),
.d(d[164]), .sel(sel));
        xtmux4b i165(.xtout(xtout[165]), .a(a[165]), .b(b[165]), .c(c[165]),
.d(d[165]), .sel(sel));
        xtmux4b i166(.xtout(xtout[166]), .a(a[166]), .b(b[166]), .c(c[166]),
.d(d[166]), .sel(sel));
        xtmux4b i167(.xtout(xtout[167]), .a(a[167]), .b(b[167]), .c(c[167]),
.d(d[167]), .sel(sel));
        xtmux4b i168(.xtout(xtout[168]), .a(a[168]), .b(b[168]), .c(c[168]),
.d(d[168]), .sel(sel));
        xtmux4b i169(.xtout(xtout[169]), .a(a[169]), .b(b[169]), .c(c[169]),
.d(d[169]), .sel(sel));
        xtmux4b i170(.xtout(xtout[170]), .a(a[170]), .b(b[170]), .c(c[170]),
.d(d[170]), .sel(sel));
        xtmux4b i171(.xtout(xtout[171]), .a(a[171]), .b(b[171]), .c(c[171]),
.d(d[171]), .sel(sel));
        xtmux4b i172(.xtout(xtout[172]), .a(a[172]), .b(b[172]), .c(c[172]),
.d(d[172]), .sel(sel));
        xtmux4b i173(.xtout(xtout[173]), .a(a[173]), .b(b[173]), .c(c[173]),
.d(d[173]), .sel(sel));
        xtmux4b i174(.xtout(xtout[174]), .a(a[174]), .b(b[174]), .c(c[174]),
.d(d[174]), .sel(sel));
        xtmux4b i175(.xtout(xtout[175]), .a(a[175]), .b(b[175]), .c(c[175]),
.d(d[175]), .sel(sel));
        xtmux4b i176(.xtout(xtout[176]), .a(a[176]), .b(b[176]), .c(c[176]),
.d(d[176]), .sel(sel));
        xtmux4b i177(.xtout(xtout[177]), .a(a[177]), .b(b[177]), .c(c[177]),
.d(d[177]), .sel(sel));
        xtmux4b i178(.xtout(xtout[178]), .a(a[178]), .b(b[178]), .c(c[178]),
.d(d[178]), .sel(sel));
        xtmux4b i179(.xtout(xtout[179]), .a(a[179]), .b(b[179]), .c(c[179]),
.d(d[179]), .sel(sel));
        xtmux4b i180(.xtout(xtout[180]), .a(a[180]), .b(b[180]), .c(c[180]),
.d(d[180]), .sel(sel));
        xtmux4b i181(.xtout(xtout[181]), .a(a[181]), .b(b[181]), .c(c[181]),
.d(d[181]), .sel(sel));
```

```
    xtmux4b i182(.xtout(xtout[182]), .a(a[182]), .b(b[182]), .c(c[182]),
.d(d[182]), .sel(sel));
    xtmux4b i183(.xtout(xtout[183]), .a(a[183]), .b(b[183]), .c(c[183]),
.d(d[183]), .sel(sel));
    xtmux4b i184(.xtout(xtout[184]), .a(a[184]), .b(b[184]), .c(c[184]),
.d(d[184]), .sel(sel));
    xtmux4b i185(.xtout(xtout[185]), .a(a[185]), .b(b[185]), .c(c[185]),
.d(d[185]), .sel(sel));
    xtmux4b i186(.xtout(xtout[186]), .a(a[186]), .b(b[186]), .c(c[186]),
.d(d[186]), .sel(sel));
    xtmux4b i187(.xtout(xtout[187]), .a(a[187]), .b(b[187]), .c(c[187]),
.d(d[187]), .sel(sel));
    xtmux4b i188(.xtout(xtout[188]), .a(a[188]), .b(b[188]), .c(c[188]),
.d(d[188]), .sel(sel));
    xtmux4b i189(.xtout(xtout[189]), .a(a[189]), .b(b[189]), .c(c[189]),
.d(d[189]), .sel(sel));
    xtmux4b i190(.xtout(xtout[190]), .a(a[190]), .b(b[190]), .c(c[190]),
.d(d[190]), .sel(sel));
    xtmux4b i191(.xtout(xtout[191]), .a(a[191]), .b(b[191]), .c(c[191]),
.d(d[191]), .sel(sel));
    xtmux4b i192(.xtout(xtout[192]), .a(a[192]), .b(b[192]), .c(c[192]),
.d(d[192]), .sel(sel));
    xtmux4b i193(.xtout(xtout[193]), .a(a[193]), .b(b[193]), .c(c[193]),
.d(d[193]), .sel(sel));
    xtmux4b i194(.xtout(xtout[194]), .a(a[194]), .b(b[194]), .c(c[194]),
.d(d[194]), .sel(sel));
    xtmux4b i195(.xtout(xtout[195]), .a(a[195]), .b(b[195]), .c(c[195]),
.d(d[195]), .sel(sel));
    xtmux4b i196(.xtout(xtout[196]), .a(a[196]), .b(b[196]), .c(c[196]),
.d(d[196]), .sel(sel));
    xtmux4b i197(.xtout(xtout[197]), .a(a[197]), .b(b[197]), .c(c[197]),
.d(d[197]), .sel(sel));
    xtmux4b i198(.xtout(xtout[198]), .a(a[198]), .b(b[198]), .c(c[198]),
.d(d[198]), .sel(sel));
    xtmux4b i199(.xtout(xtout[199]), .a(a[199]), .b(b[199]), .c(c[199]),
.d(d[199]), .sel(sel));
    xtmux4b i200(.xtout(xtout[200]), .a(a[200]), .b(b[200]), .c(c[200]),
.d(d[200]), .sel(sel));
    xtmux4b i201(.xtout(xtout[201]), .a(a[201]), .b(b[201]), .c(c[201]),
.d(d[201]), .sel(sel));
    xtmux4b i202(.xtout(xtout[202]), .a(a[202]), .b(b[202]), .c(c[202]),
.d(d[202]), .sel(sel));
    xtmux4b i203(.xtout(xtout[203]), .a(a[203]), .b(b[203]), .c(c[203]),
.d(d[203]), .sel(sel));
    xtmux4b i204(.xtout(xtout[204]), .a(a[204]), .b(b[204]), .c(c[204]),
.d(d[204]), .sel(sel));
    xtmux4b i205(.xtout(xtout[205]), .a(a[205]), .b(b[205]), .c(c[205]),
.d(d[205]), .sel(sel));
    xtmux4b i206(.xtout(xtout[206]), .a(a[206]), .b(b[206]), .c(c[206]),
.d(d[206]), .sel(sel));
    xtmux4b i207(.xtout(xtout[207]), .a(a[207]), .b(b[207]), .c(c[207]),
.d(d[207]), .sel(sel));
    xtmux4b i208(.xtout(xtout[208]), .a(a[208]), .b(b[208]), .c(c[208]),
.d(d[208]), .sel(sel));
    xtmux4b i209(.xtout(xtout[209]), .a(a[209]), .b(b[209]), .c(c[209]),
.d(d[209]), .sel(sel));
```

```
xtmux4b i210(.xtout(xtout[210]), .a(a[210]), .b(b[210]), .c(c[210]),
.d(d[210]), .sel(sel));
xtmux4b i211(.xtout(xtout[211]), .a(a[211]), .b(b[211]), .c(c[211]),
.d(d[211]), .sel(sel));
xtmux4b i212(.xtout(xtout[212]), .a(a[212]), .b(b[212]), .c(c[212]),
.d(d[212]), .sel(sel));
xtmux4b i213(.xtout(xtout[213]), .a(a[213]), .b(b[213]), .c(c[213]),
.d(d[213]), .sel(sel));
xtmux4b i214(.xtout(xtout[214]), .a(a[214]), .b(b[214]), .c(c[214]),
.d(d[214]), .sel(sel));
xtmux4b i215(.xtout(xtout[215]), .a(a[215]), .b(b[215]), .c(c[215]),
.d(d[215]), .sel(sel));
xtmux4b i216(.xtout(xtout[216]), .a(a[216]), .b(b[216]), .c(c[216]),
.d(d[216]), .sel(sel));
xtmux4b i217(.xtout(xtout[217]), .a(a[217]), .b(b[217]), .c(c[217]),
.d(d[217]), .sel(sel));
xtmux4b i218(.xtout(xtout[218]), .a(a[218]), .b(b[218]), .c(c[218]),
.d(d[218]), .sel(sel));
xtmux4b i219(.xtout(xtout[219]), .a(a[219]), .b(b[219]), .c(c[219]),
.d(d[219]), .sel(sel));
xtmux4b i220(.xtout(xtout[220]), .a(a[220]), .b(b[220]), .c(c[220]),
.d(d[220]), .sel(sel));
xtmux4b i221(.xtout(xtout[221]), .a(a[221]), .b(b[221]), .c(c[221]),
.d(d[221]), .sel(sel));
xtmux4b i222(.xtout(xtout[222]), .a(a[222]), .b(b[222]), .c(c[222]),
.d(d[222]), .sel(sel));
xtmux4b i223(.xtout(xtout[223]), .a(a[223]), .b(b[223]), .c(c[223]),
.d(d[223]), .sel(sel));
xtmux4b i224(.xtout(xtout[224]), .a(a[224]), .b(b[224]), .c(c[224]),
.d(d[224]), .sel(sel));
xtmux4b i225(.xtout(xtout[225]), .a(a[225]), .b(b[225]), .c(c[225]),
.d(d[225]), .sel(sel));
xtmux4b i226(.xtout(xtout[226]), .a(a[226]), .b(b[226]), .c(c[226]),
.d(d[226]), .sel(sel));
xtmux4b i227(.xtout(xtout[227]), .a(a[227]), .b(b[227]), .c(c[227]),
.d(d[227]), .sel(sel));
xtmux4b i228(.xtout(xtout[228]), .a(a[228]), .b(b[228]), .c(c[228]),
.d(d[228]), .sel(sel));
xtmux4b i229(.xtout(xtout[229]), .a(a[229]), .b(b[229]), .c(c[229]),
.d(d[229]), .sel(sel));
xtmux4b i230(.xtout(xtout[230]), .a(a[230]), .b(b[230]), .c(c[230]),
.d(d[230]), .sel(sel));
xtmux4b i231(.xtout(xtout[231]), .a(a[231]), .b(b[231]), .c(c[231]),
.d(d[231]), .sel(sel));
xtmux4b i232(.xtout(xtout[232]), .a(a[232]), .b(b[232]), .c(c[232]),
.d(d[232]), .sel(sel));
xtmux4b i233(.xtout(xtout[233]), .a(a[233]), .b(b[233]), .c(c[233]),
.d(d[233]), .sel(sel));
xtmux4b i234(.xtout(xtout[234]), .a(a[234]), .b(b[234]), .c(c[234]),
.d(d[234]), .sel(sel));
xtmux4b i235(.xtout(xtout[235]), .a(a[235]), .b(b[235]), .c(c[235]),
.d(d[235]), .sel(sel));
xtmux4b i236(.xtout(xtout[236]), .a(a[236]), .b(b[236]), .c(c[236]),
.d(d[236]), .sel(sel));
xtmux4b i237(.xtout(xtout[237]), .a(a[237]), .b(b[237]), .c(c[237]),
.d(d[237]), .sel(sel));
```

```
        xtmux4b i238(.xtout(xtout[238]), .a(a[238]), .b(b[238]), .c(c[238]),
    .d(d[238]), .sel(sel));
        xtmux4b i239(.xtout(xtout[239]), .a(a[239]), .b(b[239]), .c(c[239]),
    .d(d[239]), .sel(sel));
        xtmux4b i240(.xtout(xtout[240]), .a(a[240]), .b(b[240]), .c(c[240]),
    .d(d[240]), .sel(sel));
        xtmux4b i241(.xtout(xtout[241]), .a(a[241]), .b(b[241]), .c(c[241]),
    .d(d[241]), .sel(sel));
        xtmux4b i242(.xtout(xtout[242]), .a(a[242]), .b(b[242]), .c(c[242]),
    .d(d[242]), .sel(sel));
        xtmux4b i243(.xtout(xtout[243]), .a(a[243]), .b(b[243]), .c(c[243]),
    .d(d[243]), .sel(sel));
        xtmux4b i244(.xtout(xtout[244]), .a(a[244]), .b(b[244]), .c(c[244]),
    .d(d[244]), .sel(sel));
        xtmux4b i245(.xtout(xtout[245]), .a(a[245]), .b(b[245]), .c(c[245]),
    .d(d[245]), .sel(sel));
        xtmux4b i246(.xtout(xtout[246]), .a(a[246]), .b(b[246]), .c(c[246]),
    .d(d[246]), .sel(sel));
        xtmux4b i247(.xtout(xtout[247]), .a(a[247]), .b(b[247]), .c(c[247]),
    .d(d[247]), .sel(sel));
        xtmux4b i248(.xtout(xtout[248]), .a(a[248]), .b(b[248]), .c(c[248]),
    .d(d[248]), .sel(sel));
        xtmux4b i249(.xtout(xtout[249]), .a(a[249]), .b(b[249]), .c(c[249]),
    .d(d[249]), .sel(sel));
        xtmux4b i250(.xtout(xtout[250]), .a(a[250]), .b(b[250]), .c(c[250]),
    .d(d[250]), .sel(sel));
        xtmux4b i251(.xtout(xtout[251]), .a(a[251]), .b(b[251]), .c(c[251]),
    .d(d[251]), .sel(sel));
        xtmux4b i252(.xtout(xtout[252]), .a(a[252]), .b(b[252]), .c(c[252]),
    .d(d[252]), .sel(sel));
        xtmux4b i253(.xtout(xtout[253]), .a(a[253]), .b(b[253]), .c(c[253]),
    .d(d[253]), .sel(sel));
        xtmux4b i254(.xtout(xtout[254]), .a(a[254]), .b(b[254]), .c(c[254]),
    .d(d[254]), .sel(sel));
        xtmux4b i255(.xtout(xtout[255]), .a(a[255]), .b(b[255]), .c(c[255]),
    .d(d[255]), .sel(sel));
        xtmux4b i256(.xtout(xtout[256]), .a(a[256]), .b(b[256]), .c(c[256]),
    .d(d[256]), .sel(sel));
        xtmux4b i257(.xtout(xtout[257]), .a(a[257]), .b(b[257]), .c(c[257]),
    .d(d[257]), .sel(sel));
        xtmux4b i258(.xtout(xtout[258]), .a(a[258]), .b(b[258]), .c(c[258]),
    .d(d[258]), .sel(sel));
        xtmux4b i259(.xtout(xtout[259]), .a(a[259]), .b(b[259]), .c(c[259]),
    .d(d[259]), .sel(sel));
        xtmux4b i260(.xtout(xtout[260]), .a(a[260]), .b(b[260]), .c(c[260]),
    .d(d[260]), .sel(sel));
        xtmux4b i261(.xtout(xtout[261]), .a(a[261]), .b(b[261]), .c(c[261]),
    .d(d[261]), .sel(sel));
        xtmux4b i262(.xtout(xtout[262]), .a(a[262]), .b(b[262]), .c(c[262]),
    .d(d[262]), .sel(sel));
        xtmux4b i263(.xtout(xtout[263]), .a(a[263]), .b(b[263]), .c(c[263]),
    .d(d[263]), .sel(sel));
        xtmux4b i264(.xtout(xtout[264]), .a(a[264]), .b(b[264]), .c(c[264]),
    .d(d[264]), .sel(sel));
        xtmux4b i265(.xtout(xtout[265]), .a(a[265]), .b(b[265]), .c(c[265]),
    .d(d[265]), .sel(sel));
```

```
    xtmux4b i266(.xtout(xtout[266]), .a(a[266]), .b(b[266]), .c(c[266]),
.d(d[266]), .sel(sel));
    xtmux4b i267(.xtout(xtout[267]), .a(a[267]), .b(b[267]), .c(c[267]),
.d(d[267]), .sel(sel));
    xtmux4b i268(.xtout(xtout[268]), .a(a[268]), .b(b[268]), .c(c[268]),
.d(d[268]), .sel(sel));
    xtmux4b i269(.xtout(xtout[269]), .a(a[269]), .b(b[269]), .c(c[269]),
.d(d[269]), .sel(sel));
    xtmux4b i270(.xtout(xtout[270]), .a(a[270]), .b(b[270]), .c(c[270]),
.d(d[270]), .sel(sel));
    xtmux4b i271(.xtout(xtout[271]), .a(a[271]), .b(b[271]), .c(c[271]),
.d(d[271]), .sel(sel));
    xtmux4b i272(.xtout(xtout[272]), .a(a[272]), .b(b[272]), .c(c[272]),
.d(d[272]), .sel(sel));
    xtmux4b i273(.xtout(xtout[273]), .a(a[273]), .b(b[273]), .c(c[273]),
.d(d[273]), .sel(sel));
    xtmux4b i274(.xtout(xtout[274]), .a(a[274]), .b(b[274]), .c(c[274]),
.d(d[274]), .sel(sel));
    xtmux4b i275(.xtout(xtout[275]), .a(a[275]), .b(b[275]), .c(c[275]),
.d(d[275]), .sel(sel));
    xtmux4b i276(.xtout(xtout[276]), .a(a[276]), .b(b[276]), .c(c[276]),
.d(d[276]), .sel(sel));
    xtmux4b i277(.xtout(xtout[277]), .a(a[277]), .b(b[277]), .c(c[277]),
.d(d[277]), .sel(sel));
    xtmux4b i278(.xtout(xtout[278]), .a(a[278]), .b(b[278]), .c(c[278]),
.d(d[278]), .sel(sel));
    xtmux4b i279(.xtout(xtout[279]), .a(a[279]), .b(b[279]), .c(c[279]),
.d(d[279]), .sel(sel));
    xtmux4b i280(.xtout(xtout[280]), .a(a[280]), .b(b[280]), .c(c[280]),
.d(d[280]), .sel(sel));
    xtmux4b i281(.xtout(xtout[281]), .a(a[281]), .b(b[281]), .c(c[281]),
.d(d[281]), .sel(sel));
    xtmux4b i282(.xtout(xtout[282]), .a(a[282]), .b(b[282]), .c(c[282]),
.d(d[282]), .sel(sel));
    xtmux4b i283(.xtout(xtout[283]), .a(a[283]), .b(b[283]), .c(c[283]),
.d(d[283]), .sel(sel));
    xtmux4b i284(.xtout(xtout[284]), .a(a[284]), .b(b[284]), .c(c[284]),
.d(d[284]), .sel(sel));
    xtmux4b i285(.xtout(xtout[285]), .a(a[285]), .b(b[285]), .c(c[285]),
.d(d[285]), .sel(sel));
    xtmux4b i286(.xtout(xtout[286]), .a(a[286]), .b(b[286]), .c(c[286]),
.d(d[286]), .sel(sel));
    xtmux4b i287(.xtout(xtout[287]), .a(a[287]), .b(b[287]), .c(c[287]),
.d(d[287]), .sel(sel));
    xtmux4b i288(.xtout(xtout[288]), .a(a[288]), .b(b[288]), .c(c[288]),
.d(d[288]), .sel(sel));
    xtmux4b i289(.xtout(xtout[289]), .a(a[289]), .b(b[289]), .c(c[289]),
.d(d[289]), .sel(sel));
    xtmux4b i290(.xtout(xtout[290]), .a(a[290]), .b(b[290]), .c(c[290]),
.d(d[290]), .sel(sel));
    xtmux4b i291(.xtout(xtout[291]), .a(a[291]), .b(b[291]), .c(c[291]),
.d(d[291]), .sel(sel));
    xtmux4b i292(.xtout(xtout[292]), .a(a[292]), .b(b[292]), .c(c[292]),
.d(d[292]), .sel(sel));
    xtmux4b i293(.xtout(xtout[293]), .a(a[293]), .b(b[293]), .c(c[293]),
.d(d[293]), .sel(sel));
```

```
    xtmux4b i294(.xtout(xtout[294]), .a(a[294]), .b(b[294]), .c(c[294]),
.d(d[294]), .sel(sel));
    xtmux4b i295(.xtout(xtout[295]), .a(a[295]), .b(b[295]), .c(c[295]),
.d(d[295]), .sel(sel));
    xtmux4b i296(.xtout(xtout[296]), .a(a[296]), .b(b[296]), .c(c[296]),
.d(d[296]), .sel(sel));
    xtmux4b i297(.xtout(xtout[297]), .a(a[297]), .b(b[297]), .c(c[297]),
.d(d[297]), .sel(sel));
    xtmux4b i298(.xtout(xtout[298]), .a(a[298]), .b(b[298]), .c(c[298]),
.d(d[298]), .sel(sel));
    xtmux4b i299(.xtout(xtout[299]), .a(a[299]), .b(b[299]), .c(c[299]),
.d(d[299]), .sel(sel));
    xtmux4b i300(.xtout(xtout[300]), .a(a[300]), .b(b[300]), .c(c[300]),
.d(d[300]), .sel(sel));
    xtmux4b i301(.xtout(xtout[301]), .a(a[301]), .b(b[301]), .c(c[301]),
.d(d[301]), .sel(sel));
    xtmux4b i302(.xtout(xtout[302]), .a(a[302]), .b(b[302]), .c(c[302]),
.d(d[302]), .sel(sel));
    xtmux4b i303(.xtout(xtout[303]), .a(a[303]), .b(b[303]), .c(c[303]),
.d(d[303]), .sel(sel));
    xtmux4b i304(.xtout(xtout[304]), .a(a[304]), .b(b[304]), .c(c[304]),
.d(d[304]), .sel(sel));
    xtmux4b i305(.xtout(xtout[305]), .a(a[305]), .b(b[305]), .c(c[305]),
.d(d[305]), .sel(sel));
    xtmux4b i306(.xtout(xtout[306]), .a(a[306]), .b(b[306]), .c(c[306]),
.d(d[306]), .sel(sel));
    xtmux4b i307(.xtout(xtout[307]), .a(a[307]), .b(b[307]), .c(c[307]),
.d(d[307]), .sel(sel));
    xtmux4b i308(.xtout(xtout[308]), .a(a[308]), .b(b[308]), .c(c[308]),
.d(d[308]), .sel(sel));
    xtmux4b i309(.xtout(xtout[309]), .a(a[309]), .b(b[309]), .c(c[309]),
.d(d[309]), .sel(sel));
    xtmux4b i310(.xtout(xtout[310]), .a(a[310]), .b(b[310]), .c(c[310]),
.d(d[310]), .sel(sel));
    xtmux4b i311(.xtout(xtout[311]), .a(a[311]), .b(b[311]), .c(c[311]),
.d(d[311]), .sel(sel));
    xtmux4b i312(.xtout(xtout[312]), .a(a[312]), .b(b[312]), .c(c[312]),
.d(d[312]), .sel(sel));
    xtmux4b i313(.xtout(xtout[313]), .a(a[313]), .b(b[313]), .c(c[313]),
.d(d[313]), .sel(sel));
    xtmux4b i314(.xtout(xtout[314]), .a(a[314]), .b(b[314]), .c(c[314]),
.d(d[314]), .sel(sel));
    xtmux4b i315(.xtout(xtout[315]), .a(a[315]), .b(b[315]), .c(c[315]),
.d(d[315]), .sel(sel));
    xtmux4b i316(.xtout(xtout[316]), .a(a[316]), .b(b[316]), .c(c[316]),
.d(d[316]), .sel(sel));
    xtmux4b i317(.xtout(xtout[317]), .a(a[317]), .b(b[317]), .c(c[317]),
.d(d[317]), .sel(sel));
    xtmux4b i318(.xtout(xtout[318]), .a(a[318]), .b(b[318]), .c(c[318]),
.d(d[318]), .sel(sel));
    xtmux4b i319(.xtout(xtout[319]), .a(a[319]), .b(b[319]), .c(c[319]),
.d(d[319]), .sel(sel));
    xtmux4b i320(.xtout(xtout[320]), .a(a[320]), .b(b[320]), .c(c[320]),
.d(d[320]), .sel(sel));
    xtmux4b i321(.xtout(xtout[321]), .a(a[321]), .b(b[321]), .c(c[321]),
.d(d[321]), .sel(sel));
```

```
    xtmux4b i322(.xtout(xtout[322]), .a(a[322]), .b(b[322]), .c(c[322]),
.d(d[322]), .sel(sel));
    xtmux4b i323(.xtout(xtout[323]), .a(a[323]), .b(b[323]), .c(c[323]),
.d(d[323]), .sel(sel));
    xtmux4b i324(.xtout(xtout[324]), .a(a[324]), .b(b[324]), .c(c[324]),
.d(d[324]), .sel(sel));
    xtmux4b i325(.xtout(xtout[325]), .a(a[325]), .b(b[325]), .c(c[325]),
.d(d[325]), .sel(sel));
    xtmux4b i326(.xtout(xtout[326]), .a(a[326]), .b(b[326]), .c(c[326]),
.d(d[326]), .sel(sel));
    xtmux4b i327(.xtout(xtout[327]), .a(a[327]), .b(b[327]), .c(c[327]),
.d(d[327]), .sel(sel));
    xtmux4b i328(.xtout(xtout[328]), .a(a[328]), .b(b[328]), .c(c[328]),
.d(d[328]), .sel(sel));
    xtmux4b i329(.xtout(xtout[329]), .a(a[329]), .b(b[329]), .c(c[329]),
.d(d[329]), .sel(sel));
    xtmux4b i330(.xtout(xtout[330]), .a(a[330]), .b(b[330]), .c(c[330]),
.d(d[330]), .sel(sel));
    xtmux4b i331(.xtout(xtout[331]), .a(a[331]), .b(b[331]), .c(c[331]),
.d(d[331]), .sel(sel));
    xtmux4b i332(.xtout(xtout[332]), .a(a[332]), .b(b[332]), .c(c[332]),
.d(d[332]), .sel(sel));
    xtmux4b i333(.xtout(xtout[333]), .a(a[333]), .b(b[333]), .c(c[333]),
.d(d[333]), .sel(sel));
    xtmux4b i334(.xtout(xtout[334]), .a(a[334]), .b(b[334]), .c(c[334]),
.d(d[334]), .sel(sel));
    xtmux4b i335(.xtout(xtout[335]), .a(a[335]), .b(b[335]), .c(c[335]),
.d(d[335]), .sel(sel));
    xtmux4b i336(.xtout(xtout[336]), .a(a[336]), .b(b[336]), .c(c[336]),
.d(d[336]), .sel(sel));
    xtmux4b i337(.xtout(xtout[337]), .a(a[337]), .b(b[337]), .c(c[337]),
.d(d[337]), .sel(sel));
    xtmux4b i338(.xtout(xtout[338]), .a(a[338]), .b(b[338]), .c(c[338]),
.d(d[338]), .sel(sel));
    xtmux4b i339(.xtout(xtout[339]), .a(a[339]), .b(b[339]), .c(c[339]),
.d(d[339]), .sel(sel));
    xtmux4b i340(.xtout(xtout[340]), .a(a[340]), .b(b[340]), .c(c[340]),
.d(d[340]), .sel(sel));
    xtmux4b i341(.xtout(xtout[341]), .a(a[341]), .b(b[341]), .c(c[341]),
.d(d[341]), .sel(sel));
    xtmux4b i342(.xtout(xtout[342]), .a(a[342]), .b(b[342]), .c(c[342]),
.d(d[342]), .sel(sel));
    xtmux4b i343(.xtout(xtout[343]), .a(a[343]), .b(b[343]), .c(c[343]),
.d(d[343]), .sel(sel));
    xtmux4b i344(.xtout(xtout[344]), .a(a[344]), .b(b[344]), .c(c[344]),
.d(d[344]), .sel(sel));
    xtmux4b i345(.xtout(xtout[345]), .a(a[345]), .b(b[345]), .c(c[345]),
.d(d[345]), .sel(sel));
    xtmux4b i346(.xtout(xtout[346]), .a(a[346]), .b(b[346]), .c(c[346]),
.d(d[346]), .sel(sel));
    xtmux4b i347(.xtout(xtout[347]), .a(a[347]), .b(b[347]), .c(c[347]),
.d(d[347]), .sel(sel));
    xtmux4b i348(.xtout(xtout[348]), .a(a[348]), .b(b[348]), .c(c[348]),
.d(d[348]), .sel(sel));
    xtmux4b i349(.xtout(xtout[349]), .a(a[349]), .b(b[349]), .c(c[349]),
.d(d[349]), .sel(sel));
```

```
    xtmux4b i350(.xtout(xtout[350]), .a(a[350]), .b(b[350]), .c(c[350]),
.d(d[350]), .sel(sel));
    xtmux4b i351(.xtout(xtout[351]), .a(a[351]), .b(b[351]), .c(c[351]),
.d(d[351]), .sel(sel));
    xtmux4b i352(.xtout(xtout[352]), .a(a[352]), .b(b[352]), .c(c[352]),
.d(d[352]), .sel(sel));
    xtmux4b i353(.xtout(xtout[353]), .a(a[353]), .b(b[353]), .c(c[353]),
.d(d[353]), .sel(sel));
    xtmux4b i354(.xtout(xtout[354]), .a(a[354]), .b(b[354]), .c(c[354]),
.d(d[354]), .sel(sel));
    xtmux4b i355(.xtout(xtout[355]), .a(a[355]), .b(b[355]), .c(c[355]),
.d(d[355]), .sel(sel));
    xtmux4b i356(.xtout(xtout[356]), .a(a[356]), .b(b[356]), .c(c[356]),
.d(d[356]), .sel(sel));
    xtmux4b i357(.xtout(xtout[357]), .a(a[357]), .b(b[357]), .c(c[357]),
.d(d[357]), .sel(sel));
    xtmux4b i358(.xtout(xtout[358]), .a(a[358]), .b(b[358]), .c(c[358]),
.d(d[358]), .sel(sel));
    xtmux4b i359(.xtout(xtout[359]), .a(a[359]), .b(b[359]), .c(c[359]),
.d(d[359]), .sel(sel));
    xtmux4b i360(.xtout(xtout[360]), .a(a[360]), .b(b[360]), .c(c[360]),
.d(d[360]), .sel(sel));
    xtmux4b i361(.xtout(xtout[361]), .a(a[361]), .b(b[361]), .c(c[361]),
.d(d[361]), .sel(sel));
    xtmux4b i362(.xtout(xtout[362]), .a(a[362]), .b(b[362]), .c(c[362]),
.d(d[362]), .sel(sel));
    xtmux4b i363(.xtout(xtout[363]), .a(a[363]), .b(b[363]), .c(c[363]),
.d(d[363]), .sel(sel));
    xtmux4b i364(.xtout(xtout[364]), .a(a[364]), .b(b[364]), .c(c[364]),
.d(d[364]), .sel(sel));
    xtmux4b i365(.xtout(xtout[365]), .a(a[365]), .b(b[365]), .c(c[365]),
.d(d[365]), .sel(sel));
    xtmux4b i366(.xtout(xtout[366]), .a(a[366]), .b(b[366]), .c(c[366]),
.d(d[366]), .sel(sel));
    xtmux4b i367(.xtout(xtout[367]), .a(a[367]), .b(b[367]), .c(c[367]),
.d(d[367]), .sel(sel));
    xtmux4b i368(.xtout(xtout[368]), .a(a[368]), .b(b[368]), .c(c[368]),
.d(d[368]), .sel(sel));
    xtmux4b i369(.xtout(xtout[369]), .a(a[369]), .b(b[369]), .c(c[369]),
.d(d[369]), .sel(sel));
    xtmux4b i370(.xtout(xtout[370]), .a(a[370]), .b(b[370]), .c(c[370]),
.d(d[370]), .sel(sel));
    xtmux4b i371(.xtout(xtout[371]), .a(a[371]), .b(b[371]), .c(c[371]),
.d(d[371]), .sel(sel));
    xtmux4b i372(.xtout(xtout[372]), .a(a[372]), .b(b[372]), .c(c[372]),
.d(d[372]), .sel(sel));
    xtmux4b i373(.xtout(xtout[373]), .a(a[373]), .b(b[373]), .c(c[373]),
.d(d[373]), .sel(sel));
    xtmux4b i374(.xtout(xtout[374]), .a(a[374]), .b(b[374]), .c(c[374]),
.d(d[374]), .sel(sel));
    xtmux4b i375(.xtout(xtout[375]), .a(a[375]), .b(b[375]), .c(c[375]),
.d(d[375]), .sel(sel));
    xtmux4b i376(.xtout(xtout[376]), .a(a[376]), .b(b[376]), .c(c[376]),
.d(d[376]), .sel(sel));
    xtmux4b i377(.xtout(xtout[377]), .a(a[377]), .b(b[377]), .c(c[377]),
.d(d[377]), .sel(sel));
```

```
    xtmux4b i378(.xtout(xtout[378]), .a(a[378]), .b(b[378]), .c(c[378]),
.d(d[378]), .sel(sel));
    xtmux4b i379(.xtout(xtout[379]), .a(a[379]), .b(b[379]), .c(c[379]),
.d(d[379]), .sel(sel));
    xtmux4b i380(.xtout(xtout[380]), .a(a[380]), .b(b[380]), .c(c[380]),
.d(d[380]), .sel(sel));
    xtmux4b i381(.xtout(xtout[381]), .a(a[381]), .b(b[381]), .c(c[381]),
.d(d[381]), .sel(sel));
    xtmux4b i382(.xtout(xtout[382]), .a(a[382]), .b(b[382]), .c(c[382]),
.d(d[382]), .sel(sel));
    xtmux4b i383(.xtout(xtout[383]), .a(a[383]), .b(b[383]), .c(c[383]),
.d(d[383]), .sel(sel));
    xtmux4b i384(.xtout(xtout[384]), .a(a[384]), .b(b[384]), .c(c[384]),
.d(d[384]), .sel(sel));
    xtmux4b i385(.xtout(xtout[385]), .a(a[385]), .b(b[385]), .c(c[385]),
.d(d[385]), .sel(sel));
    xtmux4b i386(.xtout(xtout[386]), .a(a[386]), .b(b[386]), .c(c[386]),
.d(d[386]), .sel(sel));
    xtmux4b i387(.xtout(xtout[387]), .a(a[387]), .b(b[387]), .c(c[387]),
.d(d[387]), .sel(sel));
    xtmux4b i388(.xtout(xtout[388]), .a(a[388]), .b(b[388]), .c(c[388]),
.d(d[388]), .sel(sel));
    xtmux4b i389(.xtout(xtout[389]), .a(a[389]), .b(b[389]), .c(c[389]),
.d(d[389]), .sel(sel));
    xtmux4b i390(.xtout(xtout[390]), .a(a[390]), .b(b[390]), .c(c[390]),
.d(d[390]), .sel(sel));
    xtmux4b i391(.xtout(xtout[391]), .a(a[391]), .b(b[391]), .c(c[391]),
.d(d[391]), .sel(sel));
    xtmux4b i392(.xtout(xtout[392]), .a(a[392]), .b(b[392]), .c(c[392]),
.d(d[392]), .sel(sel));
    xtmux4b i393(.xtout(xtout[393]), .a(a[393]), .b(b[393]), .c(c[393]),
.d(d[393]), .sel(sel));
    xtmux4b i394(.xtout(xtout[394]), .a(a[394]), .b(b[394]), .c(c[394]),
.d(d[394]), .sel(sel));
    xtmux4b i395(.xtout(xtout[395]), .a(a[395]), .b(b[395]), .c(c[395]),
.d(d[395]), .sel(sel));
    xtmux4b i396(.xtout(xtout[396]), .a(a[396]), .b(b[396]), .c(c[396]),
.d(d[396]), .sel(sel));
    xtmux4b i397(.xtout(xtout[397]), .a(a[397]), .b(b[397]), .c(c[397]),
.d(d[397]), .sel(sel));
    xtmux4b i398(.xtout(xtout[398]), .a(a[398]), .b(b[398]), .c(c[398]),
.d(d[398]), .sel(sel));
    xtmux4b i399(.xtout(xtout[399]), .a(a[399]), .b(b[399]), .c(c[399]),
.d(d[399]), .sel(sel));
    xtmux4b i400(.xtout(xtout[400]), .a(a[400]), .b(b[400]), .c(c[400]),
.d(d[400]), .sel(sel));
    xtmux4b i401(.xtout(xtout[401]), .a(a[401]), .b(b[401]), .c(c[401]),
.d(d[401]), .sel(sel));
    xtmux4b i402(.xtout(xtout[402]), .a(a[402]), .b(b[402]), .c(c[402]),
.d(d[402]), .sel(sel));
    xtmux4b i403(.xtout(xtout[403]), .a(a[403]), .b(b[403]), .c(c[403]),
.d(d[403]), .sel(sel));
    xtmux4b i404(.xtout(xtout[404]), .a(a[404]), .b(b[404]), .c(c[404]),
.d(d[404]), .sel(sel));
    xtmux4b i405(.xtout(xtout[405]), .a(a[405]), .b(b[405]), .c(c[405]),
.d(d[405]), .sel(sel));
```

```
    xtmux4b i406(.xtout(xtout[406]), .a(a[406]), .b(b[406]), .c(c[406]),
.d(d[406]), .sel(sel));
    xtmux4b i407(.xtout(xtout[407]), .a(a[407]), .b(b[407]), .c(c[407]),
.d(d[407]), .sel(sel));
    xtmux4b i408(.xtout(xtout[408]), .a(a[408]), .b(b[408]), .c(c[408]),
.d(d[408]), .sel(sel));
    xtmux4b i409(.xtout(xtout[409]), .a(a[409]), .b(b[409]), .c(c[409]),
.d(d[409]), .sel(sel));
    xtmux4b i410(.xtout(xtout[410]), .a(a[410]), .b(b[410]), .c(c[410]),
.d(d[410]), .sel(sel));
    xtmux4b i411(.xtout(xtout[411]), .a(a[411]), .b(b[411]), .c(c[411]),
.d(d[411]), .sel(sel));
    xtmux4b i412(.xtout(xtout[412]), .a(a[412]), .b(b[412]), .c(c[412]),
.d(d[412]), .sel(sel));
    xtmux4b i413(.xtout(xtout[413]), .a(a[413]), .b(b[413]), .c(c[413]),
.d(d[413]), .sel(sel));
    xtmux4b i414(.xtout(xtout[414]), .a(a[414]), .b(b[414]), .c(c[414]),
.d(d[414]), .sel(sel));
    xtmux4b i415(.xtout(xtout[415]), .a(a[415]), .b(b[415]), .c(c[415]),
.d(d[415]), .sel(sel));
    xtmux4b i416(.xtout(xtout[416]), .a(a[416]), .b(b[416]), .c(c[416]),
.d(d[416]), .sel(sel));
    xtmux4b i417(.xtout(xtout[417]), .a(a[417]), .b(b[417]), .c(c[417]),
.d(d[417]), .sel(sel));
    xtmux4b i418(.xtout(xtout[418]), .a(a[418]), .b(b[418]), .c(c[418]),
.d(d[418]), .sel(sel));
    xtmux4b i419(.xtout(xtout[419]), .a(a[419]), .b(b[419]), .c(c[419]),
.d(d[419]), .sel(sel));
    xtmux4b i420(.xtout(xtout[420]), .a(a[420]), .b(b[420]), .c(c[420]),
.d(d[420]), .sel(sel));
    xtmux4b i421(.xtout(xtout[421]), .a(a[421]), .b(b[421]), .c(c[421]),
.d(d[421]), .sel(sel));
    xtmux4b i422(.xtout(xtout[422]), .a(a[422]), .b(b[422]), .c(c[422]),
.d(d[422]), .sel(sel));
    xtmux4b i423(.xtout(xtout[423]), .a(a[423]), .b(b[423]), .c(c[423]),
.d(d[423]), .sel(sel));
    xtmux4b i424(.xtout(xtout[424]), .a(a[424]), .b(b[424]), .c(c[424]),
.d(d[424]), .sel(sel));
    xtmux4b i425(.xtout(xtout[425]), .a(a[425]), .b(b[425]), .c(c[425]),
.d(d[425]), .sel(sel));
    xtmux4b i426(.xtout(xtout[426]), .a(a[426]), .b(b[426]), .c(c[426]),
.d(d[426]), .sel(sel));
    xtmux4b i427(.xtout(xtout[427]), .a(a[427]), .b(b[427]), .c(c[427]),
.d(d[427]), .sel(sel));
    xtmux4b i428(.xtout(xtout[428]), .a(a[428]), .b(b[428]), .c(c[428]),
.d(d[428]), .sel(sel));
    xtmux4b i429(.xtout(xtout[429]), .a(a[429]), .b(b[429]), .c(c[429]),
.d(d[429]), .sel(sel));
    xtmux4b i430(.xtout(xtout[430]), .a(a[430]), .b(b[430]), .c(c[430]),
.d(d[430]), .sel(sel));
    xtmux4b i431(.xtout(xtout[431]), .a(a[431]), .b(b[431]), .c(c[431]),
.d(d[431]), .sel(sel));
    xtmux4b i432(.xtout(xtout[432]), .a(a[432]), .b(b[432]), .c(c[432]),
.d(d[432]), .sel(sel));
    xtmux4b i433(.xtout(xtout[433]), .a(a[433]), .b(b[433]), .c(c[433]),
.d(d[433]), .sel(sel));
```

```
    xtmux4b i434(.xtout(xtout[434]), .a(a[434]), .b(b[434]), .c(c[434]),
.d(d[434]), .sel(sel));
    xtmux4b i435(.xtout(xtout[435]), .a(a[435]), .b(b[435]), .c(c[435]),
.d(d[435]), .sel(sel));
    xtmux4b i436(.xtout(xtout[436]), .a(a[436]), .b(b[436]), .c(c[436]),
.d(d[436]), .sel(sel));
    xtmux4b i437(.xtout(xtout[437]), .a(a[437]), .b(b[437]), .c(c[437]),
.d(d[437]), .sel(sel));
    xtmux4b i438(.xtout(xtout[438]), .a(a[438]), .b(b[438]), .c(c[438]),
.d(d[438]), .sel(sel));
    xtmux4b i439(.xtout(xtout[439]), .a(a[439]), .b(b[439]), .c(c[439]),
.d(d[439]), .sel(sel));
    xtmux4b i440(.xtout(xtout[440]), .a(a[440]), .b(b[440]), .c(c[440]),
.d(d[440]), .sel(sel));
    xtmux4b i441(.xtout(xtout[441]), .a(a[441]), .b(b[441]), .c(c[441]),
.d(d[441]), .sel(sel));
    xtmux4b i442(.xtout(xtout[442]), .a(a[442]), .b(b[442]), .c(c[442]),
.d(d[442]), .sel(sel));
    xtmux4b i443(.xtout(xtout[443]), .a(a[443]), .b(b[443]), .c(c[443]),
.d(d[443]), .sel(sel));
    xtmux4b i444(.xtout(xtout[444]), .a(a[444]), .b(b[444]), .c(c[444]),
.d(d[444]), .sel(sel));
    xtmux4b i445(.xtout(xtout[445]), .a(a[445]), .b(b[445]), .c(c[445]),
.d(d[445]), .sel(sel));
    xtmux4b i446(.xtout(xtout[446]), .a(a[446]), .b(b[446]), .c(c[446]),
.d(d[446]), .sel(sel));
    xtmux4b i447(.xtout(xtout[447]), .a(a[447]), .b(b[447]), .c(c[447]),
.d(d[447]), .sel(sel));
    xtmux4b i448(.xtout(xtout[448]), .a(a[448]), .b(b[448]), .c(c[448]),
.d(d[448]), .sel(sel));
    xtmux4b i449(.xtout(xtout[449]), .a(a[449]), .b(b[449]), .c(c[449]),
.d(d[449]), .sel(sel));
    xtmux4b i450(.xtout(xtout[450]), .a(a[450]), .b(b[450]), .c(c[450]),
.d(d[450]), .sel(sel));
    xtmux4b i451(.xtout(xtout[451]), .a(a[451]), .b(b[451]), .c(c[451]),
.d(d[451]), .sel(sel));
    xtmux4b i452(.xtout(xtout[452]), .a(a[452]), .b(b[452]), .c(c[452]),
.d(d[452]), .sel(sel));
    xtmux4b i453(.xtout(xtout[453]), .a(a[453]), .b(b[453]), .c(c[453]),
.d(d[453]), .sel(sel));
    xtmux4b i454(.xtout(xtout[454]), .a(a[454]), .b(b[454]), .c(c[454]),
.d(d[454]), .sel(sel));
    xtmux4b i455(.xtout(xtout[455]), .a(a[455]), .b(b[455]), .c(c[455]),
.d(d[455]), .sel(sel));
    xtmux4b i456(.xtout(xtout[456]), .a(a[456]), .b(b[456]), .c(c[456]),
.d(d[456]), .sel(sel));
    xtmux4b i457(.xtout(xtout[457]), .a(a[457]), .b(b[457]), .c(c[457]),
.d(d[457]), .sel(sel));
    xtmux4b i458(.xtout(xtout[458]), .a(a[458]), .b(b[458]), .c(c[458]),
.d(d[458]), .sel(sel));
    xtmux4b i459(.xtout(xtout[459]), .a(a[459]), .b(b[459]), .c(c[459]),
.d(d[459]), .sel(sel));
    xtmux4b i460(.xtout(xtout[460]), .a(a[460]), .b(b[460]), .c(c[460]),
.d(d[460]), .sel(sel));
    xtmux4b i461(.xtout(xtout[461]), .a(a[461]), .b(b[461]), .c(c[461]),
.d(d[461]), .sel(sel));
```

```
    xtmux4b i462(.xtout(xtout[462]), .a(a[462]), .b(b[462]), .c(c[462]),
.d(d[462]), .sel(sel));
    xtmux4b i463(.xtout(xtout[463]), .a(a[463]), .b(b[463]), .c(c[463]),
.d(d[463]), .sel(sel));
    xtmux4b i464(.xtout(xtout[464]), .a(a[464]), .b(b[464]), .c(c[464]),
.d(d[464]), .sel(sel));
    xtmux4b i465(.xtout(xtout[465]), .a(a[465]), .b(b[465]), .c(c[465]),
.d(d[465]), .sel(sel));
    xtmux4b i466(.xtout(xtout[466]), .a(a[466]), .b(b[466]), .c(c[466]),
.d(d[466]), .sel(sel));
    xtmux4b i467(.xtout(xtout[467]), .a(a[467]), .b(b[467]), .c(c[467]),
.d(d[467]), .sel(sel));
    xtmux4b i468(.xtout(xtout[468]), .a(a[468]), .b(b[468]), .c(c[468]),
.d(d[468]), .sel(sel));
    xtmux4b i469(.xtout(xtout[469]), .a(a[469]), .b(b[469]), .c(c[469]),
.d(d[469]), .sel(sel));
    xtmux4b i470(.xtout(xtout[470]), .a(a[470]), .b(b[470]), .c(c[470]),
.d(d[470]), .sel(sel));
    xtmux4b i471(.xtout(xtout[471]), .a(a[471]), .b(b[471]), .c(c[471]),
.d(d[471]), .sel(sel));
    xtmux4b i472(.xtout(xtout[472]), .a(a[472]), .b(b[472]), .c(c[472]),
.d(d[472]), .sel(sel));
    xtmux4b i473(.xtout(xtout[473]), .a(a[473]), .b(b[473]), .c(c[473]),
.d(d[473]), .sel(sel));
    xtmux4b i474(.xtout(xtout[474]), .a(a[474]), .b(b[474]), .c(c[474]),
.d(d[474]), .sel(sel));
    xtmux4b i475(.xtout(xtout[475]), .a(a[475]), .b(b[475]), .c(c[475]),
.d(d[475]), .sel(sel));
    xtmux4b i476(.xtout(xtout[476]), .a(a[476]), .b(b[476]), .c(c[476]),
.d(d[476]), .sel(sel));
    xtmux4b i477(.xtout(xtout[477]), .a(a[477]), .b(b[477]), .c(c[477]),
.d(d[477]), .sel(sel));
    xtmux4b i478(.xtout(xtout[478]), .a(a[478]), .b(b[478]), .c(c[478]),
.d(d[478]), .sel(sel));
    xtmux4b i479(.xtout(xtout[479]), .a(a[479]), .b(b[479]), .c(c[479]),
.d(d[479]), .sel(sel));
    xtmux4b i480(.xtout(xtout[480]), .a(a[480]), .b(b[480]), .c(c[480]),
.d(d[480]), .sel(sel));
    xtmux4b i481(.xtout(xtout[481]), .a(a[481]), .b(b[481]), .c(c[481]),
.d(d[481]), .sel(sel));
    xtmux4b i482(.xtout(xtout[482]), .a(a[482]), .b(b[482]), .c(c[482]),
.d(d[482]), .sel(sel));
    xtmux4b i483(.xtout(xtout[483]), .a(a[483]), .b(b[483]), .c(c[483]),
.d(d[483]), .sel(sel));
    xtmux4b i484(.xtout(xtout[484]), .a(a[484]), .b(b[484]), .c(c[484]),
.d(d[484]), .sel(sel));
    xtmux4b i485(.xtout(xtout[485]), .a(a[485]), .b(b[485]), .c(c[485]),
.d(d[485]), .sel(sel));
    xtmux4b i486(.xtout(xtout[486]), .a(a[486]), .b(b[486]), .c(c[486]),
.d(d[486]), .sel(sel));
    xtmux4b i487(.xtout(xtout[487]), .a(a[487]), .b(b[487]), .c(c[487]),
.d(d[487]), .sel(sel));
    xtmux4b i488(.xtout(xtout[488]), .a(a[488]), .b(b[488]), .c(c[488]),
.d(d[488]), .sel(sel));
    xtmux4b i489(.xtout(xtout[489]), .a(a[489]), .b(b[489]), .c(c[489]),
.d(d[489]), .sel(sel));
```

```
    xtmux4b i490(.xtout(xtout[490]), .a(a[490]), .b(b[490]), .c(c[490]),
.d(d[490]), .sel(sel));
    xtmux4b i491(.xtout(xtout[491]), .a(a[491]), .b(b[491]), .c(c[491]),
.d(d[491]), .sel(sel));
    xtmux4b i492(.xtout(xtout[492]), .a(a[492]), .b(b[492]), .c(c[492]),
.d(d[492]), .sel(sel));
    xtmux4b i493(.xtout(xtout[493]), .a(a[493]), .b(b[493]), .c(c[493]),
.d(d[493]), .sel(sel));
    xtmux4b i494(.xtout(xtout[494]), .a(a[494]), .b(b[494]), .c(c[494]),
.d(d[494]), .sel(sel));
    xtmux4b i495(.xtout(xtout[495]), .a(a[495]), .b(b[495]), .c(c[495]),
.d(d[495]), .sel(sel));
    xtmux4b i496(.xtout(xtout[496]), .a(a[496]), .b(b[496]), .c(c[496]),
.d(d[496]), .sel(sel));
    xtmux4b i497(.xtout(xtout[497]), .a(a[497]), .b(b[497]), .c(c[497]),
.d(d[497]), .sel(sel));
    xtmux4b i498(.xtout(xtout[498]), .a(a[498]), .b(b[498]), .c(c[498]),
.d(d[498]), .sel(sel));
    xtmux4b i499(.xtout(xtout[499]), .a(a[499]), .b(b[499]), .c(c[499]),
.d(d[499]), .sel(sel));
    xtmux4b i500(.xtout(xtout[500]), .a(a[500]), .b(b[500]), .c(c[500]),
.d(d[500]), .sel(sel));
    xtmux4b i501(.xtout(xtout[501]), .a(a[501]), .b(b[501]), .c(c[501]),
.d(d[501]), .sel(sel));
    xtmux4b i502(.xtout(xtout[502]), .a(a[502]), .b(b[502]), .c(c[502]),
.d(d[502]), .sel(sel));
    xtmux4b i503(.xtout(xtout[503]), .a(a[503]), .b(b[503]), .c(c[503]),
.d(d[503]), .sel(sel));
    xtmux4b i504(.xtout(xtout[504]), .a(a[504]), .b(b[504]), .c(c[504]),
.d(d[504]), .sel(sel));
    xtmux4b i505(.xtout(xtout[505]), .a(a[505]), .b(b[505]), .c(c[505]),
.d(d[505]), .sel(sel));
    xtmux4b i506(.xtout(xtout[506]), .a(a[506]), .b(b[506]), .c(c[506]),
.d(d[506]), .sel(sel));
    xtmux4b i507(.xtout(xtout[507]), .a(a[507]), .b(b[507]), .c(c[507]),
.d(d[507]), .sel(sel));
    xtmux4b i508(.xtout(xtout[508]), .a(a[508]), .b(b[508]), .c(c[508]),
.d(d[508]), .sel(sel));
    xtmux4b i509(.xtout(xtout[509]), .a(a[509]), .b(b[509]), .c(c[509]),
.d(d[509]), .sel(sel));
    xtmux4b i510(.xtout(xtout[510]), .a(a[510]), .b(b[510]), .c(c[510]),
.d(d[510]), .sel(sel));
    xtmux4b i511(.xtout(xtout[511]), .a(a[511]), .b(b[511]), .c(c[511]),
.d(d[511]), .sel(sel));
    xtmux4b i512(.xtout(xtout[512]), .a(a[512]), .b(b[512]), .c(c[512]),
.d(d[512]), .sel(sel));
    xtmux4b i513(.xtout(xtout[513]), .a(a[513]), .b(b[513]), .c(c[513]),
.d(d[513]), .sel(sel));
    xtmux4b i514(.xtout(xtout[514]), .a(a[514]), .b(b[514]), .c(c[514]),
.d(d[514]), .sel(sel));
    xtmux4b i515(.xtout(xtout[515]), .a(a[515]), .b(b[515]), .c(c[515]),
.d(d[515]), .sel(sel));
    xtmux4b i516(.xtout(xtout[516]), .a(a[516]), .b(b[516]), .c(c[516]),
.d(d[516]), .sel(sel));
    xtmux4b i517(.xtout(xtout[517]), .a(a[517]), .b(b[517]), .c(c[517]),
.d(d[517]), .sel(sel));
```

```
    xtmux4b i518(.xtout(xtout[518]), .a(a[518]), .b(b[518]), .c(c[518]),
.d(d[518]), .sel(sel));
    xtmux4b i519(.xtout(xtout[519]), .a(a[519]), .b(b[519]), .c(c[519]),
.d(d[519]), .sel(sel));
    xtmux4b i520(.xtout(xtout[520]), .a(a[520]), .b(b[520]), .c(c[520]),
.d(d[520]), .sel(sel));
    xtmux4b i521(.xtout(xtout[521]), .a(a[521]), .b(b[521]), .c(c[521]),
.d(d[521]), .sel(sel));
    xtmux4b i522(.xtout(xtout[522]), .a(a[522]), .b(b[522]), .c(c[522]),
.d(d[522]), .sel(sel));
    xtmux4b i523(.xtout(xtout[523]), .a(a[523]), .b(b[523]), .c(c[523]),
.d(d[523]), .sel(sel));
    xtmux4b i524(.xtout(xtout[524]), .a(a[524]), .b(b[524]), .c(c[524]),
.d(d[524]), .sel(sel));
    xtmux4b i525(.xtout(xtout[525]), .a(a[525]), .b(b[525]), .c(c[525]),
.d(d[525]), .sel(sel));
    xtmux4b i526(.xtout(xtout[526]), .a(a[526]), .b(b[526]), .c(c[526]),
.d(d[526]), .sel(sel));
    xtmux4b i527(.xtout(xtout[527]), .a(a[527]), .b(b[527]), .c(c[527]),
.d(d[527]), .sel(sel));
    xtmux4b i528(.xtout(xtout[528]), .a(a[528]), .b(b[528]), .c(c[528]),
.d(d[528]), .sel(sel));
    xtmux4b i529(.xtout(xtout[529]), .a(a[529]), .b(b[529]), .c(c[529]),
.d(d[529]), .sel(sel));
    xtmux4b i530(.xtout(xtout[530]), .a(a[530]), .b(b[530]), .c(c[530]),
.d(d[530]), .sel(sel));
    xtmux4b i531(.xtout(xtout[531]), .a(a[531]), .b(b[531]), .c(c[531]),
.d(d[531]), .sel(sel));
    xtmux4b i532(.xtout(xtout[532]), .a(a[532]), .b(b[532]), .c(c[532]),
.d(d[532]), .sel(sel));
    xtmux4b i533(.xtout(xtout[533]), .a(a[533]), .b(b[533]), .c(c[533]),
.d(d[533]), .sel(sel));
    xtmux4b i534(.xtout(xtout[534]), .a(a[534]), .b(b[534]), .c(c[534]),
.d(d[534]), .sel(sel));
    xtmux4b i535(.xtout(xtout[535]), .a(a[535]), .b(b[535]), .c(c[535]),
.d(d[535]), .sel(sel));
    xtmux4b i536(.xtout(xtout[536]), .a(a[536]), .b(b[536]), .c(c[536]),
.d(d[536]), .sel(sel));
    xtmux4b i537(.xtout(xtout[537]), .a(a[537]), .b(b[537]), .c(c[537]),
.d(d[537]), .sel(sel));
    xtmux4b i538(.xtout(xtout[538]), .a(a[538]), .b(b[538]), .c(c[538]),
.d(d[538]), .sel(sel));
    xtmux4b i539(.xtout(xtout[539]), .a(a[539]), .b(b[539]), .c(c[539]),
.d(d[539]), .sel(sel));
    xtmux4b i540(.xtout(xtout[540]), .a(a[540]), .b(b[540]), .c(c[540]),
.d(d[540]), .sel(sel));
    xtmux4b i541(.xtout(xtout[541]), .a(a[541]), .b(b[541]), .c(c[541]),
.d(d[541]), .sel(sel));
    xtmux4b i542(.xtout(xtout[542]), .a(a[542]), .b(b[542]), .c(c[542]),
.d(d[542]), .sel(sel));
    xtmux4b i543(.xtout(xtout[543]), .a(a[543]), .b(b[543]), .c(c[543]),
.d(d[543]), .sel(sel));
    xtmux4b i544(.xtout(xtout[544]), .a(a[544]), .b(b[544]), .c(c[544]),
.d(d[544]), .sel(sel));
    xtmux4b i545(.xtout(xtout[545]), .a(a[545]), .b(b[545]), .c(c[545]),
.d(d[545]), .sel(sel));
```

```
    xtmux4b i546(.xtout(xtout[546]), .a(a[546]), .b(b[546]), .c(c[546]),
.d(d[546]), .sel(sel));
    xtmux4b i547(.xtout(xtout[547]), .a(a[547]), .b(b[547]), .c(c[547]),
.d(d[547]), .sel(sel));
    xtmux4b i548(.xtout(xtout[548]), .a(a[548]), .b(b[548]), .c(c[548]),
.d(d[548]), .sel(sel));
    xtmux4b i549(.xtout(xtout[549]), .a(a[549]), .b(b[549]), .c(c[549]),
.d(d[549]), .sel(sel));
    xtmux4b i550(.xtout(xtout[550]), .a(a[550]), .b(b[550]), .c(c[550]),
.d(d[550]), .sel(sel));
    xtmux4b i551(.xtout(xtout[551]), .a(a[551]), .b(b[551]), .c(c[551]),
.d(d[551]), .sel(sel));
    xtmux4b i552(.xtout(xtout[552]), .a(a[552]), .b(b[552]), .c(c[552]),
.d(d[552]), .sel(sel));
    xtmux4b i553(.xtout(xtout[553]), .a(a[553]), .b(b[553]), .c(c[553]),
.d(d[553]), .sel(sel));
    xtmux4b i554(.xtout(xtout[554]), .a(a[554]), .b(b[554]), .c(c[554]),
.d(d[554]), .sel(sel));
    xtmux4b i555(.xtout(xtout[555]), .a(a[555]), .b(b[555]), .c(c[555]),
.d(d[555]), .sel(sel));
    xtmux4b i556(.xtout(xtout[556]), .a(a[556]), .b(b[556]), .c(c[556]),
.d(d[556]), .sel(sel));
    xtmux4b i557(.xtout(xtout[557]), .a(a[557]), .b(b[557]), .c(c[557]),
.d(d[557]), .sel(sel));
    xtmux4b i558(.xtout(xtout[558]), .a(a[558]), .b(b[558]), .c(c[558]),
.d(d[558]), .sel(sel));
    xtmux4b i559(.xtout(xtout[559]), .a(a[559]), .b(b[559]), .c(c[559]),
.d(d[559]), .sel(sel));
    xtmux4b i560(.xtout(xtout[560]), .a(a[560]), .b(b[560]), .c(c[560]),
.d(d[560]), .sel(sel));
    xtmux4b i561(.xtout(xtout[561]), .a(a[561]), .b(b[561]), .c(c[561]),
.d(d[561]), .sel(sel));
    xtmux4b i562(.xtout(xtout[562]), .a(a[562]), .b(b[562]), .c(c[562]),
.d(d[562]), .sel(sel));
    xtmux4b i563(.xtout(xtout[563]), .a(a[563]), .b(b[563]), .c(c[563]),
.d(d[563]), .sel(sel));
    xtmux4b i564(.xtout(xtout[564]), .a(a[564]), .b(b[564]), .c(c[564]),
.d(d[564]), .sel(sel));
    xtmux4b i565(.xtout(xtout[565]), .a(a[565]), .b(b[565]), .c(c[565]),
.d(d[565]), .sel(sel));
    xtmux4b i566(.xtout(xtout[566]), .a(a[566]), .b(b[566]), .c(c[566]),
.d(d[566]), .sel(sel));
    xtmux4b i567(.xtout(xtout[567]), .a(a[567]), .b(b[567]), .c(c[567]),
.d(d[567]), .sel(sel));
    xtmux4b i568(.xtout(xtout[568]), .a(a[568]), .b(b[568]), .c(c[568]),
.d(d[568]), .sel(sel));
    xtmux4b i569(.xtout(xtout[569]), .a(a[569]), .b(b[569]), .c(c[569]),
.d(d[569]), .sel(sel));
    xtmux4b i570(.xtout(xtout[570]), .a(a[570]), .b(b[570]), .c(c[570]),
.d(d[570]), .sel(sel));
    xtmux4b i571(.xtout(xtout[571]), .a(a[571]), .b(b[571]), .c(c[571]),
.d(d[571]), .sel(sel));
    xtmux4b i572(.xtout(xtout[572]), .a(a[572]), .b(b[572]), .c(c[572]),
.d(d[572]), .sel(sel));
    xtmux4b i573(.xtout(xtout[573]), .a(a[573]), .b(b[573]), .c(c[573]),
.d(d[573]), .sel(sel));
```

```
    xtmux4b i574(.xtout(xtout[574]), .a(a[574]), .b(b[574]), .c(c[574]),
.d(d[574]), .sel(sel));
    xtmux4b i575(.xtout(xtout[575]), .a(a[575]), .b(b[575]), .c(c[575]),
.d(d[575]), .sel(sel));
    xtmux4b i576(.xtout(xtout[576]), .a(a[576]), .b(b[576]), .c(c[576]),
.d(d[576]), .sel(sel));
    xtmux4b i577(.xtout(xtout[577]), .a(a[577]), .b(b[577]), .c(c[577]),
.d(d[577]), .sel(sel));
    xtmux4b i578(.xtout(xtout[578]), .a(a[578]), .b(b[578]), .c(c[578]),
.d(d[578]), .sel(sel));
    xtmux4b i579(.xtout(xtout[579]), .a(a[579]), .b(b[579]), .c(c[579]),
.d(d[579]), .sel(sel));
    xtmux4b i580(.xtout(xtout[580]), .a(a[580]), .b(b[580]), .c(c[580]),
.d(d[580]), .sel(sel));
    xtmux4b i581(.xtout(xtout[581]), .a(a[581]), .b(b[581]), .c(c[581]),
.d(d[581]), .sel(sel));
    xtmux4b i582(.xtout(xtout[582]), .a(a[582]), .b(b[582]), .c(c[582]),
.d(d[582]), .sel(sel));
    xtmux4b i583(.xtout(xtout[583]), .a(a[583]), .b(b[583]), .c(c[583]),
.d(d[583]), .sel(sel));
    xtmux4b i584(.xtout(xtout[584]), .a(a[584]), .b(b[584]), .c(c[584]),
.d(d[584]), .sel(sel));
    xtmux4b i585(.xtout(xtout[585]), .a(a[585]), .b(b[585]), .c(c[585]),
.d(d[585]), .sel(sel));
    xtmux4b i586(.xtout(xtout[586]), .a(a[586]), .b(b[586]), .c(c[586]),
.d(d[586]), .sel(sel));
    xtmux4b i587(.xtout(xtout[587]), .a(a[587]), .b(b[587]), .c(c[587]),
.d(d[587]), .sel(sel));
    xtmux4b i588(.xtout(xtout[588]), .a(a[588]), .b(b[588]), .c(c[588]),
.d(d[588]), .sel(sel));
    xtmux4b i589(.xtout(xtout[589]), .a(a[589]), .b(b[589]), .c(c[589]),
.d(d[589]), .sel(sel));
    xtmux4b i590(.xtout(xtout[590]), .a(a[590]), .b(b[590]), .c(c[590]),
.d(d[590]), .sel(sel));
    xtmux4b i591(.xtout(xtout[591]), .a(a[591]), .b(b[591]), .c(c[591]),
.d(d[591]), .sel(sel));
    xtmux4b i592(.xtout(xtout[592]), .a(a[592]), .b(b[592]), .c(c[592]),
.d(d[592]), .sel(sel));
    xtmux4b i593(.xtout(xtout[593]), .a(a[593]), .b(b[593]), .c(c[593]),
.d(d[593]), .sel(sel));
    xtmux4b i594(.xtout(xtout[594]), .a(a[594]), .b(b[594]), .c(c[594]),
.d(d[594]), .sel(sel));
    xtmux4b i595(.xtout(xtout[595]), .a(a[595]), .b(b[595]), .c(c[595]),
.d(d[595]), .sel(sel));
    xtmux4b i596(.xtout(xtout[596]), .a(a[596]), .b(b[596]), .c(c[596]),
.d(d[596]), .sel(sel));
    xtmux4b i597(.xtout(xtout[597]), .a(a[597]), .b(b[597]), .c(c[597]),
.d(d[597]), .sel(sel));
    xtmux4b i598(.xtout(xtout[598]), .a(a[598]), .b(b[598]), .c(c[598]),
.d(d[598]), .sel(sel));
    xtmux4b i599(.xtout(xtout[599]), .a(a[599]), .b(b[599]), .c(c[599]),
.d(d[599]), .sel(sel));
    xtmux4b i600(.xtout(xtout[600]), .a(a[600]), .b(b[600]), .c(c[600]),
.d(d[600]), .sel(sel));
    xtmux4b i601(.xtout(xtout[601]), .a(a[601]), .b(b[601]), .c(c[601]),
.d(d[601]), .sel(sel));
```

```
xtmux4b i602(.xtout(xtout[602]), .a(a[602]), .b(b[602]), .c(c[602]),
.d(d[602]), .sel(sel));
xtmux4b i603(.xtout(xtout[603]), .a(a[603]), .b(b[603]), .c(c[603]),
.d(d[603]), .sel(sel));
xtmux4b i604(.xtout(xtout[604]), .a(a[604]), .b(b[604]), .c(c[604]),
.d(d[604]), .sel(sel));
xtmux4b i605(.xtout(xtout[605]), .a(a[605]), .b(b[605]), .c(c[605]),
.d(d[605]), .sel(sel));
xtmux4b i606(.xtout(xtout[606]), .a(a[606]), .b(b[606]), .c(c[606]),
.d(d[606]), .sel(sel));
xtmux4b i607(.xtout(xtout[607]), .a(a[607]), .b(b[607]), .c(c[607]),
.d(d[607]), .sel(sel));
xtmux4b i608(.xtout(xtout[608]), .a(a[608]), .b(b[608]), .c(c[608]),
.d(d[608]), .sel(sel));
xtmux4b i609(.xtout(xtout[609]), .a(a[609]), .b(b[609]), .c(c[609]),
.d(d[609]), .sel(sel));
xtmux4b i610(.xtout(xtout[610]), .a(a[610]), .b(b[610]), .c(c[610]),
.d(d[610]), .sel(sel));
xtmux4b i611(.xtout(xtout[611]), .a(a[611]), .b(b[611]), .c(c[611]),
.d(d[611]), .sel(sel));
xtmux4b i612(.xtout(xtout[612]), .a(a[612]), .b(b[612]), .c(c[612]),
.d(d[612]), .sel(sel));
xtmux4b i613(.xtout(xtout[613]), .a(a[613]), .b(b[613]), .c(c[613]),
.d(d[613]), .sel(sel));
xtmux4b i614(.xtout(xtout[614]), .a(a[614]), .b(b[614]), .c(c[614]),
.d(d[614]), .sel(sel));
xtmux4b i615(.xtout(xtout[615]), .a(a[615]), .b(b[615]), .c(c[615]),
.d(d[615]), .sel(sel));
xtmux4b i616(.xtout(xtout[616]), .a(a[616]), .b(b[616]), .c(c[616]),
.d(d[616]), .sel(sel));
xtmux4b i617(.xtout(xtout[617]), .a(a[617]), .b(b[617]), .c(c[617]),
.d(d[617]), .sel(sel));
xtmux4b i618(.xtout(xtout[618]), .a(a[618]), .b(b[618]), .c(c[618]),
.d(d[618]), .sel(sel));
xtmux4b i619(.xtout(xtout[619]), .a(a[619]), .b(b[619]), .c(c[619]),
.d(d[619]), .sel(sel));
xtmux4b i620(.xtout(xtout[620]), .a(a[620]), .b(b[620]), .c(c[620]),
.d(d[620]), .sel(sel));
xtmux4b i621(.xtout(xtout[621]), .a(a[621]), .b(b[621]), .c(c[621]),
.d(d[621]), .sel(sel));
xtmux4b i622(.xtout(xtout[622]), .a(a[622]), .b(b[622]), .c(c[622]),
.d(d[622]), .sel(sel));
xtmux4b i623(.xtout(xtout[623]), .a(a[623]), .b(b[623]), .c(c[623]),
.d(d[623]), .sel(sel));
xtmux4b i624(.xtout(xtout[624]), .a(a[624]), .b(b[624]), .c(c[624]),
.d(d[624]), .sel(sel));
xtmux4b i625(.xtout(xtout[625]), .a(a[625]), .b(b[625]), .c(c[625]),
.d(d[625]), .sel(sel));
xtmux4b i626(.xtout(xtout[626]), .a(a[626]), .b(b[626]), .c(c[626]),
.d(d[626]), .sel(sel));
xtmux4b i627(.xtout(xtout[627]), .a(a[627]), .b(b[627]), .c(c[627]),
.d(d[627]), .sel(sel));
xtmux4b i628(.xtout(xtout[628]), .a(a[628]), .b(b[628]), .c(c[628]),
.d(d[628]), .sel(sel));
xtmux4b i629(.xtout(xtout[629]), .a(a[629]), .b(b[629]), .c(c[629]),
.d(d[629]), .sel(sel));
```

```
    xtmux4b i630(.xtout(xtout[630]), .a(a[630]), .b(b[630]), .c(c[630]),
.d(d[630]), .sel(sel));
    xtmux4b i631(.xtout(xtout[631]), .a(a[631]), .b(b[631]), .c(c[631]),
.d(d[631]), .sel(sel));
    xtmux4b i632(.xtout(xtout[632]), .a(a[632]), .b(b[632]), .c(c[632]),
.d(d[632]), .sel(sel));
    xtmux4b i633(.xtout(xtout[633]), .a(a[633]), .b(b[633]), .c(c[633]),
.d(d[633]), .sel(sel));
    xtmux4b i634(.xtout(xtout[634]), .a(a[634]), .b(b[634]), .c(c[634]),
.d(d[634]), .sel(sel));
    xtmux4b i635(.xtout(xtout[635]), .a(a[635]), .b(b[635]), .c(c[635]),
.d(d[635]), .sel(sel));
    xtmux4b i636(.xtout(xtout[636]), .a(a[636]), .b(b[636]), .c(c[636]),
.d(d[636]), .sel(sel));
    xtmux4b i637(.xtout(xtout[637]), .a(a[637]), .b(b[637]), .c(c[637]),
.d(d[637]), .sel(sel));
    xtmux4b i638(.xtout(xtout[638]), .a(a[638]), .b(b[638]), .c(c[638]),
.d(d[638]), .sel(sel));
    xtmux4b i639(.xtout(xtout[639]), .a(a[639]), .b(b[639]), .c(c[639]),
.d(d[639]), .sel(sel));
    xtmux4b i640(.xtout(xtout[640]), .a(a[640]), .b(b[640]), .c(c[640]),
.d(d[640]), .sel(sel));
    xtmux4b i641(.xtout(xtout[641]), .a(a[641]), .b(b[641]), .c(c[641]),
.d(d[641]), .sel(sel));
    xtmux4b i642(.xtout(xtout[642]), .a(a[642]), .b(b[642]), .c(c[642]),
.d(d[642]), .sel(sel));
    xtmux4b i643(.xtout(xtout[643]), .a(a[643]), .b(b[643]), .c(c[643]),
.d(d[643]), .sel(sel));
    xtmux4b i644(.xtout(xtout[644]), .a(a[644]), .b(b[644]), .c(c[644]),
.d(d[644]), .sel(sel));
    xtmux4b i645(.xtout(xtout[645]), .a(a[645]), .b(b[645]), .c(c[645]),
.d(d[645]), .sel(sel));
    xtmux4b i646(.xtout(xtout[646]), .a(a[646]), .b(b[646]), .c(c[646]),
.d(d[646]), .sel(sel));
    xtmux4b i647(.xtout(xtout[647]), .a(a[647]), .b(b[647]), .c(c[647]),
.d(d[647]), .sel(sel));
    xtmux4b i648(.xtout(xtout[648]), .a(a[648]), .b(b[648]), .c(c[648]),
.d(d[648]), .sel(sel));
    xtmux4b i649(.xtout(xtout[649]), .a(a[649]), .b(b[649]), .c(c[649]),
.d(d[649]), .sel(sel));
    xtmux4b i650(.xtout(xtout[650]), .a(a[650]), .b(b[650]), .c(c[650]),
.d(d[650]), .sel(sel));
    xtmux4b i651(.xtout(xtout[651]), .a(a[651]), .b(b[651]), .c(c[651]),
.d(d[651]), .sel(sel));
    xtmux4b i652(.xtout(xtout[652]), .a(a[652]), .b(b[652]), .c(c[652]),
.d(d[652]), .sel(sel));
    xtmux4b i653(.xtout(xtout[653]), .a(a[653]), .b(b[653]), .c(c[653]),
.d(d[653]), .sel(sel));
    xtmux4b i654(.xtout(xtout[654]), .a(a[654]), .b(b[654]), .c(c[654]),
.d(d[654]), .sel(sel));
    xtmux4b i655(.xtout(xtout[655]), .a(a[655]), .b(b[655]), .c(c[655]),
.d(d[655]), .sel(sel));
    xtmux4b i656(.xtout(xtout[656]), .a(a[656]), .b(b[656]), .c(c[656]),
.d(d[656]), .sel(sel));
    xtmux4b i657(.xtout(xtout[657]), .a(a[657]), .b(b[657]), .c(c[657]),
.d(d[657]), .sel(sel));
```

```
    xtmux4b i658(.xtout(xtout[658]), .a(a[658]), .b(b[658]), .c(c[658]),
.d(d[658]), .sel(sel));
    xtmux4b i659(.xtout(xtout[659]), .a(a[659]), .b(b[659]), .c(c[659]),
.d(d[659]), .sel(sel));
    xtmux4b i660(.xtout(xtout[660]), .a(a[660]), .b(b[660]), .c(c[660]),
.d(d[660]), .sel(sel));
    xtmux4b i661(.xtout(xtout[661]), .a(a[661]), .b(b[661]), .c(c[661]),
.d(d[661]), .sel(sel));
    xtmux4b i662(.xtout(xtout[662]), .a(a[662]), .b(b[662]), .c(c[662]),
.d(d[662]), .sel(sel));
    xtmux4b i663(.xtout(xtout[663]), .a(a[663]), .b(b[663]), .c(c[663]),
.d(d[663]), .sel(sel));
    xtmux4b i664(.xtout(xtout[664]), .a(a[664]), .b(b[664]), .c(c[664]),
.d(d[664]), .sel(sel));
    xtmux4b i665(.xtout(xtout[665]), .a(a[665]), .b(b[665]), .c(c[665]),
.d(d[665]), .sel(sel));
    xtmux4b i666(.xtout(xtout[666]), .a(a[666]), .b(b[666]), .c(c[666]),
.d(d[666]), .sel(sel));
    xtmux4b i667(.xtout(xtout[667]), .a(a[667]), .b(b[667]), .c(c[667]),
.d(d[667]), .sel(sel));
    xtmux4b i668(.xtout(xtout[668]), .a(a[668]), .b(b[668]), .c(c[668]),
.d(d[668]), .sel(sel));
    xtmux4b i669(.xtout(xtout[669]), .a(a[669]), .b(b[669]), .c(c[669]),
.d(d[669]), .sel(sel));
    xtmux4b i670(.xtout(xtout[670]), .a(a[670]), .b(b[670]), .c(c[670]),
.d(d[670]), .sel(sel));
    xtmux4b i671(.xtout(xtout[671]), .a(a[671]), .b(b[671]), .c(c[671]),
.d(d[671]), .sel(sel));
    xtmux4b i672(.xtout(xtout[672]), .a(a[672]), .b(b[672]), .c(c[672]),
.d(d[672]), .sel(sel));
    xtmux4b i673(.xtout(xtout[673]), .a(a[673]), .b(b[673]), .c(c[673]),
.d(d[673]), .sel(sel));
    xtmux4b i674(.xtout(xtout[674]), .a(a[674]), .b(b[674]), .c(c[674]),
.d(d[674]), .sel(sel));
    xtmux4b i675(.xtout(xtout[675]), .a(a[675]), .b(b[675]), .c(c[675]),
.d(d[675]), .sel(sel));
    xtmux4b i676(.xtout(xtout[676]), .a(a[676]), .b(b[676]), .c(c[676]),
.d(d[676]), .sel(sel));
    xtmux4b i677(.xtout(xtout[677]), .a(a[677]), .b(b[677]), .c(c[677]),
.d(d[677]), .sel(sel));
    xtmux4b i678(.xtout(xtout[678]), .a(a[678]), .b(b[678]), .c(c[678]),
.d(d[678]), .sel(sel));
    xtmux4b i679(.xtout(xtout[679]), .a(a[679]), .b(b[679]), .c(c[679]),
.d(d[679]), .sel(sel));
    xtmux4b i680(.xtout(xtout[680]), .a(a[680]), .b(b[680]), .c(c[680]),
.d(d[680]), .sel(sel));
    xtmux4b i681(.xtout(xtout[681]), .a(a[681]), .b(b[681]), .c(c[681]),
.d(d[681]), .sel(sel));
    xtmux4b i682(.xtout(xtout[682]), .a(a[682]), .b(b[682]), .c(c[682]),
.d(d[682]), .sel(sel));
    xtmux4b i683(.xtout(xtout[683]), .a(a[683]), .b(b[683]), .c(c[683]),
.d(d[683]), .sel(sel));
    xtmux4b i684(.xtout(xtout[684]), .a(a[684]), .b(b[684]), .c(c[684]),
.d(d[684]), .sel(sel));
    xtmux4b i685(.xtout(xtout[685]), .a(a[685]), .b(b[685]), .c(c[685]),
.d(d[685]), .sel(sel));
```

```
    xtmux4b i686(.xtout(xtout[686]), .a(a[686]), .b(b[686]), .c(c[686]),
.d(d[686]), .sel(sel));
    xtmux4b i687(.xtout(xtout[687]), .a(a[687]), .b(b[687]), .c(c[687]),
.d(d[687]), .sel(sel));
    xtmux4b i688(.xtout(xtout[688]), .a(a[688]), .b(b[688]), .c(c[688]),
.d(d[688]), .sel(sel));
    xtmux4b i689(.xtout(xtout[689]), .a(a[689]), .b(b[689]), .c(c[689]),
.d(d[689]), .sel(sel));
    xtmux4b i690(.xtout(xtout[690]), .a(a[690]), .b(b[690]), .c(c[690]),
.d(d[690]), .sel(sel));
    xtmux4b i691(.xtout(xtout[691]), .a(a[691]), .b(b[691]), .c(c[691]),
.d(d[691]), .sel(sel));
    xtmux4b i692(.xtout(xtout[692]), .a(a[692]), .b(b[692]), .c(c[692]),
.d(d[692]), .sel(sel));
    xtmux4b i693(.xtout(xtout[693]), .a(a[693]), .b(b[693]), .c(c[693]),
.d(d[693]), .sel(sel));
    xtmux4b i694(.xtout(xtout[694]), .a(a[694]), .b(b[694]), .c(c[694]),
.d(d[694]), .sel(sel));
    xtmux4b i695(.xtout(xtout[695]), .a(a[695]), .b(b[695]), .c(c[695]),
.d(d[695]), .sel(sel));
    xtmux4b i696(.xtout(xtout[696]), .a(a[696]), .b(b[696]), .c(c[696]),
.d(d[696]), .sel(sel));
    xtmux4b i697(.xtout(xtout[697]), .a(a[697]), .b(b[697]), .c(c[697]),
.d(d[697]), .sel(sel));
    xtmux4b i698(.xtout(xtout[698]), .a(a[698]), .b(b[698]), .c(c[698]),
.d(d[698]), .sel(sel));
    xtmux4b i699(.xtout(xtout[699]), .a(a[699]), .b(b[699]), .c(c[699]),
.d(d[699]), .sel(sel));
    xtmux4b i700(.xtout(xtout[700]), .a(a[700]), .b(b[700]), .c(c[700]),
.d(d[700]), .sel(sel));
    xtmux4b i701(.xtout(xtout[701]), .a(a[701]), .b(b[701]), .c(c[701]),
.d(d[701]), .sel(sel));
    xtmux4b i702(.xtout(xtout[702]), .a(a[702]), .b(b[702]), .c(c[702]),
.d(d[702]), .sel(sel));
    xtmux4b i703(.xtout(xtout[703]), .a(a[703]), .b(b[703]), .c(c[703]),
.d(d[703]), .sel(sel));
    xtmux4b i704(.xtout(xtout[704]), .a(a[704]), .b(b[704]), .c(c[704]),
.d(d[704]), .sel(sel));
    xtmux4b i705(.xtout(xtout[705]), .a(a[705]), .b(b[705]), .c(c[705]),
.d(d[705]), .sel(sel));
    xtmux4b i706(.xtout(xtout[706]), .a(a[706]), .b(b[706]), .c(c[706]),
.d(d[706]), .sel(sel));
    xtmux4b i707(.xtout(xtout[707]), .a(a[707]), .b(b[707]), .c(c[707]),
.d(d[707]), .sel(sel));
    xtmux4b i708(.xtout(xtout[708]), .a(a[708]), .b(b[708]), .c(c[708]),
.d(d[708]), .sel(sel));
    xtmux4b i709(.xtout(xtout[709]), .a(a[709]), .b(b[709]), .c(c[709]),
.d(d[709]), .sel(sel));
    xtmux4b i710(.xtout(xtout[710]), .a(a[710]), .b(b[710]), .c(c[710]),
.d(d[710]), .sel(sel));
    xtmux4b i711(.xtout(xtout[711]), .a(a[711]), .b(b[711]), .c(c[711]),
.d(d[711]), .sel(sel));
    xtmux4b i712(.xtout(xtout[712]), .a(a[712]), .b(b[712]), .c(c[712]),
.d(d[712]), .sel(sel));
    xtmux4b i713(.xtout(xtout[713]), .a(a[713]), .b(b[713]), .c(c[713]),
.d(d[713]), .sel(sel));
```

```
    xtmux4b i714(.xtout(xtout[714]), .a(a[714]), .b(b[714]), .c(c[714]),
.d(d[714]), .sel(sel));
    xtmux4b i715(.xtout(xtout[715]), .a(a[715]), .b(b[715]), .c(c[715]),
.d(d[715]), .sel(sel));
    xtmux4b i716(.xtout(xtout[716]), .a(a[716]), .b(b[716]), .c(c[716]),
.d(d[716]), .sel(sel));
    xtmux4b i717(.xtout(xtout[717]), .a(a[717]), .b(b[717]), .c(c[717]),
.d(d[717]), .sel(sel));
    xtmux4b i718(.xtout(xtout[718]), .a(a[718]), .b(b[718]), .c(c[718]),
.d(d[718]), .sel(sel));
    xtmux4b i719(.xtout(xtout[719]), .a(a[719]), .b(b[719]), .c(c[719]),
.d(d[719]), .sel(sel));
    xtmux4b i720(.xtout(xtout[720]), .a(a[720]), .b(b[720]), .c(c[720]),
.d(d[720]), .sel(sel));
    xtmux4b i721(.xtout(xtout[721]), .a(a[721]), .b(b[721]), .c(c[721]),
.d(d[721]), .sel(sel));
    xtmux4b i722(.xtout(xtout[722]), .a(a[722]), .b(b[722]), .c(c[722]),
.d(d[722]), .sel(sel));
    xtmux4b i723(.xtout(xtout[723]), .a(a[723]), .b(b[723]), .c(c[723]),
.d(d[723]), .sel(sel));
    xtmux4b i724(.xtout(xtout[724]), .a(a[724]), .b(b[724]), .c(c[724]),
.d(d[724]), .sel(sel));
    xtmux4b i725(.xtout(xtout[725]), .a(a[725]), .b(b[725]), .c(c[725]),
.d(d[725]), .sel(sel));
    xtmux4b i726(.xtout(xtout[726]), .a(a[726]), .b(b[726]), .c(c[726]),
.d(d[726]), .sel(sel));
    xtmux4b i727(.xtout(xtout[727]), .a(a[727]), .b(b[727]), .c(c[727]),
.d(d[727]), .sel(sel));
    xtmux4b i728(.xtout(xtout[728]), .a(a[728]), .b(b[728]), .c(c[728]),
.d(d[728]), .sel(sel));
    xtmux4b i729(.xtout(xtout[729]), .a(a[729]), .b(b[729]), .c(c[729]),
.d(d[729]), .sel(sel));
    xtmux4b i730(.xtout(xtout[730]), .a(a[730]), .b(b[730]), .c(c[730]),
.d(d[730]), .sel(sel));
    xtmux4b i731(.xtout(xtout[731]), .a(a[731]), .b(b[731]), .c(c[731]),
.d(d[731]), .sel(sel));
    xtmux4b i732(.xtout(xtout[732]), .a(a[732]), .b(b[732]), .c(c[732]),
.d(d[732]), .sel(sel));
    xtmux4b i733(.xtout(xtout[733]), .a(a[733]), .b(b[733]), .c(c[733]),
.d(d[733]), .sel(sel));
    xtmux4b i734(.xtout(xtout[734]), .a(a[734]), .b(b[734]), .c(c[734]),
.d(d[734]), .sel(sel));
    xtmux4b i735(.xtout(xtout[735]), .a(a[735]), .b(b[735]), .c(c[735]),
.d(d[735]), .sel(sel));
    xtmux4b i736(.xtout(xtout[736]), .a(a[736]), .b(b[736]), .c(c[736]),
.d(d[736]), .sel(sel));
    xtmux4b i737(.xtout(xtout[737]), .a(a[737]), .b(b[737]), .c(c[737]),
.d(d[737]), .sel(sel));
    xtmux4b i738(.xtout(xtout[738]), .a(a[738]), .b(b[738]), .c(c[738]),
.d(d[738]), .sel(sel));
    xtmux4b i739(.xtout(xtout[739]), .a(a[739]), .b(b[739]), .c(c[739]),
.d(d[739]), .sel(sel));
    xtmux4b i740(.xtout(xtout[740]), .a(a[740]), .b(b[740]), .c(c[740]),
.d(d[740]), .sel(sel));
    xtmux4b i741(.xtout(xtout[741]), .a(a[741]), .b(b[741]), .c(c[741]),
.d(d[741]), .sel(sel));
```

```
    xtmux4b i742(.xtout(xtout[742]), .a(a[742]), .b(b[742]), .c(c[742]),
.d(d[742]), .sel(sel));
    xtmux4b i743(.xtout(xtout[743]), .a(a[743]), .b(b[743]), .c(c[743]),
.d(d[743]), .sel(sel));
    xtmux4b i744(.xtout(xtout[744]), .a(a[744]), .b(b[744]), .c(c[744]),
.d(d[744]), .sel(sel));
    xtmux4b i745(.xtout(xtout[745]), .a(a[745]), .b(b[745]), .c(c[745]),
.d(d[745]), .sel(sel));
    xtmux4b i746(.xtout(xtout[746]), .a(a[746]), .b(b[746]), .c(c[746]),
.d(d[746]), .sel(sel));
    xtmux4b i747(.xtout(xtout[747]), .a(a[747]), .b(b[747]), .c(c[747]),
.d(d[747]), .sel(sel));
    xtmux4b i748(.xtout(xtout[748]), .a(a[748]), .b(b[748]), .c(c[748]),
.d(d[748]), .sel(sel));
    xtmux4b i749(.xtout(xtout[749]), .a(a[749]), .b(b[749]), .c(c[749]),
.d(d[749]), .sel(sel));
    xtmux4b i750(.xtout(xtout[750]), .a(a[750]), .b(b[750]), .c(c[750]),
.d(d[750]), .sel(sel));
    xtmux4b i751(.xtout(xtout[751]), .a(a[751]), .b(b[751]), .c(c[751]),
.d(d[751]), .sel(sel));
    xtmux4b i752(.xtout(xtout[752]), .a(a[752]), .b(b[752]), .c(c[752]),
.d(d[752]), .sel(sel));
    xtmux4b i753(.xtout(xtout[753]), .a(a[753]), .b(b[753]), .c(c[753]),
.d(d[753]), .sel(sel));
    xtmux4b i754(.xtout(xtout[754]), .a(a[754]), .b(b[754]), .c(c[754]),
.d(d[754]), .sel(sel));
    xtmux4b i755(.xtout(xtout[755]), .a(a[755]), .b(b[755]), .c(c[755]),
.d(d[755]), .sel(sel));
    xtmux4b i756(.xtout(xtout[756]), .a(a[756]), .b(b[756]), .c(c[756]),
.d(d[756]), .sel(sel));
    xtmux4b i757(.xtout(xtout[757]), .a(a[757]), .b(b[757]), .c(c[757]),
.d(d[757]), .sel(sel));
    xtmux4b i758(.xtout(xtout[758]), .a(a[758]), .b(b[758]), .c(c[758]),
.d(d[758]), .sel(sel));
    xtmux4b i759(.xtout(xtout[759]), .a(a[759]), .b(b[759]), .c(c[759]),
.d(d[759]), .sel(sel));
    xtmux4b i760(.xtout(xtout[760]), .a(a[760]), .b(b[760]), .c(c[760]),
.d(d[760]), .sel(sel));
    xtmux4b i761(.xtout(xtout[761]), .a(a[761]), .b(b[761]), .c(c[761]),
.d(d[761]), .sel(sel));
    xtmux4b i762(.xtout(xtout[762]), .a(a[762]), .b(b[762]), .c(c[762]),
.d(d[762]), .sel(sel));
    xtmux4b i763(.xtout(xtout[763]), .a(a[763]), .b(b[763]), .c(c[763]),
.d(d[763]), .sel(sel));
    xtmux4b i764(.xtout(xtout[764]), .a(a[764]), .b(b[764]), .c(c[764]),
.d(d[764]), .sel(sel));
    xtmux4b i765(.xtout(xtout[765]), .a(a[765]), .b(b[765]), .c(c[765]),
.d(d[765]), .sel(sel));
    xtmux4b i766(.xtout(xtout[766]), .a(a[766]), .b(b[766]), .c(c[766]),
.d(d[766]), .sel(sel));
    xtmux4b i767(.xtout(xtout[767]), .a(a[767]), .b(b[767]), .c(c[767]),
.d(d[767]), .sel(sel));
    xtmux4b i768(.xtout(xtout[768]), .a(a[768]), .b(b[768]), .c(c[768]),
.d(d[768]), .sel(sel));
    xtmux4b i769(.xtout(xtout[769]), .a(a[769]), .b(b[769]), .c(c[769]),
.d(d[769]), .sel(sel));
```

```
    xtmux4b i770(.xtout(xtout[770]), .a(a[770]), .b(b[770]), .c(c[770]),
.d(d[770]), .sel(sel));
    xtmux4b i771(.xtout(xtout[771]), .a(a[771]), .b(b[771]), .c(c[771]),
.d(d[771]), .sel(sel));
    xtmux4b i772(.xtout(xtout[772]), .a(a[772]), .b(b[772]), .c(c[772]),
.d(d[772]), .sel(sel));
    xtmux4b i773(.xtout(xtout[773]), .a(a[773]), .b(b[773]), .c(c[773]),
.d(d[773]), .sel(sel));
    xtmux4b i774(.xtout(xtout[774]), .a(a[774]), .b(b[774]), .c(c[774]),
.d(d[774]), .sel(sel));
    xtmux4b i775(.xtout(xtout[775]), .a(a[775]), .b(b[775]), .c(c[775]),
.d(d[775]), .sel(sel));
    xtmux4b i776(.xtout(xtout[776]), .a(a[776]), .b(b[776]), .c(c[776]),
.d(d[776]), .sel(sel));
    xtmux4b i777(.xtout(xtout[777]), .a(a[777]), .b(b[777]), .c(c[777]),
.d(d[777]), .sel(sel));
    xtmux4b i778(.xtout(xtout[778]), .a(a[778]), .b(b[778]), .c(c[778]),
.d(d[778]), .sel(sel));
    xtmux4b i779(.xtout(xtout[779]), .a(a[779]), .b(b[779]), .c(c[779]),
.d(d[779]), .sel(sel));
    xtmux4b i780(.xtout(xtout[780]), .a(a[780]), .b(b[780]), .c(c[780]),
.d(d[780]), .sel(sel));
    xtmux4b i781(.xtout(xtout[781]), .a(a[781]), .b(b[781]), .c(c[781]),
.d(d[781]), .sel(sel));
    xtmux4b i782(.xtout(xtout[782]), .a(a[782]), .b(b[782]), .c(c[782]),
.d(d[782]), .sel(sel));
    xtmux4b i783(.xtout(xtout[783]), .a(a[783]), .b(b[783]), .c(c[783]),
.d(d[783]), .sel(sel));
    xtmux4b i784(.xtout(xtout[784]), .a(a[784]), .b(b[784]), .c(c[784]),
.d(d[784]), .sel(sel));
    xtmux4b i785(.xtout(xtout[785]), .a(a[785]), .b(b[785]), .c(c[785]),
.d(d[785]), .sel(sel));
    xtmux4b i786(.xtout(xtout[786]), .a(a[786]), .b(b[786]), .c(c[786]),
.d(d[786]), .sel(sel));
    xtmux4b i787(.xtout(xtout[787]), .a(a[787]), .b(b[787]), .c(c[787]),
.d(d[787]), .sel(sel));
    xtmux4b i788(.xtout(xtout[788]), .a(a[788]), .b(b[788]), .c(c[788]),
.d(d[788]), .sel(sel));
    xtmux4b i789(.xtout(xtout[789]), .a(a[789]), .b(b[789]), .c(c[789]),
.d(d[789]), .sel(sel));
    xtmux4b i790(.xtout(xtout[790]), .a(a[790]), .b(b[790]), .c(c[790]),
.d(d[790]), .sel(sel));
    xtmux4b i791(.xtout(xtout[791]), .a(a[791]), .b(b[791]), .c(c[791]),
.d(d[791]), .sel(sel));
    xtmux4b i792(.xtout(xtout[792]), .a(a[792]), .b(b[792]), .c(c[792]),
.d(d[792]), .sel(sel));
    xtmux4b i793(.xtout(xtout[793]), .a(a[793]), .b(b[793]), .c(c[793]),
.d(d[793]), .sel(sel));
    xtmux4b i794(.xtout(xtout[794]), .a(a[794]), .b(b[794]), .c(c[794]),
.d(d[794]), .sel(sel));
    xtmux4b i795(.xtout(xtout[795]), .a(a[795]), .b(b[795]), .c(c[795]),
.d(d[795]), .sel(sel));
    xtmux4b i796(.xtout(xtout[796]), .a(a[796]), .b(b[796]), .c(c[796]),
.d(d[796]), .sel(sel));
    xtmux4b i797(.xtout(xtout[797]), .a(a[797]), .b(b[797]), .c(c[797]),
.d(d[797]), .sel(sel));
```

```
    xtmux4b i798(.xtout(xtout[798]), .a(a[798]), .b(b[798]), .c(c[798]),
.d(d[798]), .sel(sel));
    xtmux4b i799(.xtout(xtout[799]), .a(a[799]), .b(b[799]), .c(c[799]),
.d(d[799]), .sel(sel));
    xtmux4b i800(.xtout(xtout[800]), .a(a[800]), .b(b[800]), .c(c[800]),
.d(d[800]), .sel(sel));
    xtmux4b i801(.xtout(xtout[801]), .a(a[801]), .b(b[801]), .c(c[801]),
.d(d[801]), .sel(sel));
    xtmux4b i802(.xtout(xtout[802]), .a(a[802]), .b(b[802]), .c(c[802]),
.d(d[802]), .sel(sel));
    xtmux4b i803(.xtout(xtout[803]), .a(a[803]), .b(b[803]), .c(c[803]),
.d(d[803]), .sel(sel));
    xtmux4b i804(.xtout(xtout[804]), .a(a[804]), .b(b[804]), .c(c[804]),
.d(d[804]), .sel(sel));
    xtmux4b i805(.xtout(xtout[805]), .a(a[805]), .b(b[805]), .c(c[805]),
.d(d[805]), .sel(sel));
    xtmux4b i806(.xtout(xtout[806]), .a(a[806]), .b(b[806]), .c(c[806]),
.d(d[806]), .sel(sel));
    xtmux4b i807(.xtout(xtout[807]), .a(a[807]), .b(b[807]), .c(c[807]),
.d(d[807]), .sel(sel));
    xtmux4b i808(.xtout(xtout[808]), .a(a[808]), .b(b[808]), .c(c[808]),
.d(d[808]), .sel(sel));
    xtmux4b i809(.xtout(xtout[809]), .a(a[809]), .b(b[809]), .c(c[809]),
.d(d[809]), .sel(sel));
    xtmux4b i810(.xtout(xtout[810]), .a(a[810]), .b(b[810]), .c(c[810]),
.d(d[810]), .sel(sel));
    xtmux4b i811(.xtout(xtout[811]), .a(a[811]), .b(b[811]), .c(c[811]),
.d(d[811]), .sel(sel));
    xtmux4b i812(.xtout(xtout[812]), .a(a[812]), .b(b[812]), .c(c[812]),
.d(d[812]), .sel(sel));
    xtmux4b i813(.xtout(xtout[813]), .a(a[813]), .b(b[813]), .c(c[813]),
.d(d[813]), .sel(sel));
    xtmux4b i814(.xtout(xtout[814]), .a(a[814]), .b(b[814]), .c(c[814]),
.d(d[814]), .sel(sel));
    xtmux4b i815(.xtout(xtout[815]), .a(a[815]), .b(b[815]), .c(c[815]),
.d(d[815]), .sel(sel));
    xtmux4b i816(.xtout(xtout[816]), .a(a[816]), .b(b[816]), .c(c[816]),
.d(d[816]), .sel(sel));
    xtmux4b i817(.xtout(xtout[817]), .a(a[817]), .b(b[817]), .c(c[817]),
.d(d[817]), .sel(sel));
    xtmux4b i818(.xtout(xtout[818]), .a(a[818]), .b(b[818]), .c(c[818]),
.d(d[818]), .sel(sel));
    xtmux4b i819(.xtout(xtout[819]), .a(a[819]), .b(b[819]), .c(c[819]),
.d(d[819]), .sel(sel));
    xtmux4b i820(.xtout(xtout[820]), .a(a[820]), .b(b[820]), .c(c[820]),
.d(d[820]), .sel(sel));
    xtmux4b i821(.xtout(xtout[821]), .a(a[821]), .b(b[821]), .c(c[821]),
.d(d[821]), .sel(sel));
    xtmux4b i822(.xtout(xtout[822]), .a(a[822]), .b(b[822]), .c(c[822]),
.d(d[822]), .sel(sel));
    xtmux4b i823(.xtout(xtout[823]), .a(a[823]), .b(b[823]), .c(c[823]),
.d(d[823]), .sel(sel));
    xtmux4b i824(.xtout(xtout[824]), .a(a[824]), .b(b[824]), .c(c[824]),
.d(d[824]), .sel(sel));
    xtmux4b i825(.xtout(xtout[825]), .a(a[825]), .b(b[825]), .c(c[825]),
.d(d[825]), .sel(sel));
```

```
    xtmux4b i826(.xtout(xtout[826]), .a(a[826]), .b(b[826]), .c(c[826]),
.d(d[826]), .sel(sel));
    xtmux4b i827(.xtout(xtout[827]), .a(a[827]), .b(b[827]), .c(c[827]),
.d(d[827]), .sel(sel));
    xtmux4b i828(.xtout(xtout[828]), .a(a[828]), .b(b[828]), .c(c[828]),
.d(d[828]), .sel(sel));
    xtmux4b i829(.xtout(xtout[829]), .a(a[829]), .b(b[829]), .c(c[829]),
.d(d[829]), .sel(sel));
    xtmux4b i830(.xtout(xtout[830]), .a(a[830]), .b(b[830]), .c(c[830]),
.d(d[830]), .sel(sel));
    xtmux4b i831(.xtout(xtout[831]), .a(a[831]), .b(b[831]), .c(c[831]),
.d(d[831]), .sel(sel));
    xtmux4b i832(.xtout(xtout[832]), .a(a[832]), .b(b[832]), .c(c[832]),
.d(d[832]), .sel(sel));
    xtmux4b i833(.xtout(xtout[833]), .a(a[833]), .b(b[833]), .c(c[833]),
.d(d[833]), .sel(sel));
    xtmux4b i834(.xtout(xtout[834]), .a(a[834]), .b(b[834]), .c(c[834]),
.d(d[834]), .sel(sel));
    xtmux4b i835(.xtout(xtout[835]), .a(a[835]), .b(b[835]), .c(c[835]),
.d(d[835]), .sel(sel));
    xtmux4b i836(.xtout(xtout[836]), .a(a[836]), .b(b[836]), .c(c[836]),
.d(d[836]), .sel(sel));
    xtmux4b i837(.xtout(xtout[837]), .a(a[837]), .b(b[837]), .c(c[837]),
.d(d[837]), .sel(sel));
    xtmux4b i838(.xtout(xtout[838]), .a(a[838]), .b(b[838]), .c(c[838]),
.d(d[838]), .sel(sel));
    xtmux4b i839(.xtout(xtout[839]), .a(a[839]), .b(b[839]), .c(c[839]),
.d(d[839]), .sel(sel));
    xtmux4b i840(.xtout(xtout[840]), .a(a[840]), .b(b[840]), .c(c[840]),
.d(d[840]), .sel(sel));
    xtmux4b i841(.xtout(xtout[841]), .a(a[841]), .b(b[841]), .c(c[841]),
.d(d[841]), .sel(sel));
    xtmux4b i842(.xtout(xtout[842]), .a(a[842]), .b(b[842]), .c(c[842]),
.d(d[842]), .sel(sel));
    xtmux4b i843(.xtout(xtout[843]), .a(a[843]), .b(b[843]), .c(c[843]),
.d(d[843]), .sel(sel));
    xtmux4b i844(.xtout(xtout[844]), .a(a[844]), .b(b[844]), .c(c[844]),
.d(d[844]), .sel(sel));
    xtmux4b i845(.xtout(xtout[845]), .a(a[845]), .b(b[845]), .c(c[845]),
.d(d[845]), .sel(sel));
    xtmux4b i846(.xtout(xtout[846]), .a(a[846]), .b(b[846]), .c(c[846]),
.d(d[846]), .sel(sel));
    xtmux4b i847(.xtout(xtout[847]), .a(a[847]), .b(b[847]), .c(c[847]),
.d(d[847]), .sel(sel));
    xtmux4b i848(.xtout(xtout[848]), .a(a[848]), .b(b[848]), .c(c[848]),
.d(d[848]), .sel(sel));
    xtmux4b i849(.xtout(xtout[849]), .a(a[849]), .b(b[849]), .c(c[849]),
.d(d[849]), .sel(sel));
    xtmux4b i850(.xtout(xtout[850]), .a(a[850]), .b(b[850]), .c(c[850]),
.d(d[850]), .sel(sel));
    xtmux4b i851(.xtout(xtout[851]), .a(a[851]), .b(b[851]), .c(c[851]),
.d(d[851]), .sel(sel));
    xtmux4b i852(.xtout(xtout[852]), .a(a[852]), .b(b[852]), .c(c[852]),
.d(d[852]), .sel(sel));
    xtmux4b i853(.xtout(xtout[853]), .a(a[853]), .b(b[853]), .c(c[853]),
.d(d[853]), .sel(sel));
```

```
    xtmux4b i854(.xtout(xtout[854]), .a(a[854]), .b(b[854]), .c(c[854]),
.d(d[854]), .sel(sel));
    xtmux4b i855(.xtout(xtout[855]), .a(a[855]), .b(b[855]), .c(c[855]),
.d(d[855]), .sel(sel));
    xtmux4b i856(.xtout(xtout[856]), .a(a[856]), .b(b[856]), .c(c[856]),
.d(d[856]), .sel(sel));
    xtmux4b i857(.xtout(xtout[857]), .a(a[857]), .b(b[857]), .c(c[857]),
.d(d[857]), .sel(sel));
    xtmux4b i858(.xtout(xtout[858]), .a(a[858]), .b(b[858]), .c(c[858]),
.d(d[858]), .sel(sel));
    xtmux4b i859(.xtout(xtout[859]), .a(a[859]), .b(b[859]), .c(c[859]),
.d(d[859]), .sel(sel));
    xtmux4b i860(.xtout(xtout[860]), .a(a[860]), .b(b[860]), .c(c[860]),
.d(d[860]), .sel(sel));
    xtmux4b i861(.xtout(xtout[861]), .a(a[861]), .b(b[861]), .c(c[861]),
.d(d[861]), .sel(sel));
    xtmux4b i862(.xtout(xtout[862]), .a(a[862]), .b(b[862]), .c(c[862]),
.d(d[862]), .sel(sel));
    xtmux4b i863(.xtout(xtout[863]), .a(a[863]), .b(b[863]), .c(c[863]),
.d(d[863]), .sel(sel));
    xtmux4b i864(.xtout(xtout[864]), .a(a[864]), .b(b[864]), .c(c[864]),
.d(d[864]), .sel(sel));
    xtmux4b i865(.xtout(xtout[865]), .a(a[865]), .b(b[865]), .c(c[865]),
.d(d[865]), .sel(sel));
    xtmux4b i866(.xtout(xtout[866]), .a(a[866]), .b(b[866]), .c(c[866]),
.d(d[866]), .sel(sel));
    xtmux4b i867(.xtout(xtout[867]), .a(a[867]), .b(b[867]), .c(c[867]),
.d(d[867]), .sel(sel));
    xtmux4b i868(.xtout(xtout[868]), .a(a[868]), .b(b[868]), .c(c[868]),
.d(d[868]), .sel(sel));
    xtmux4b i869(.xtout(xtout[869]), .a(a[869]), .b(b[869]), .c(c[869]),
.d(d[869]), .sel(sel));
    xtmux4b i870(.xtout(xtout[870]), .a(a[870]), .b(b[870]), .c(c[870]),
.d(d[870]), .sel(sel));
    xtmux4b i871(.xtout(xtout[871]), .a(a[871]), .b(b[871]), .c(c[871]),
.d(d[871]), .sel(sel));
    xtmux4b i872(.xtout(xtout[872]), .a(a[872]), .b(b[872]), .c(c[872]),
.d(d[872]), .sel(sel));
    xtmux4b i873(.xtout(xtout[873]), .a(a[873]), .b(b[873]), .c(c[873]),
.d(d[873]), .sel(sel));
    xtmux4b i874(.xtout(xtout[874]), .a(a[874]), .b(b[874]), .c(c[874]),
.d(d[874]), .sel(sel));
    xtmux4b i875(.xtout(xtout[875]), .a(a[875]), .b(b[875]), .c(c[875]),
.d(d[875]), .sel(sel));
    xtmux4b i876(.xtout(xtout[876]), .a(a[876]), .b(b[876]), .c(c[876]),
.d(d[876]), .sel(sel));
    xtmux4b i877(.xtout(xtout[877]), .a(a[877]), .b(b[877]), .c(c[877]),
.d(d[877]), .sel(sel));
    xtmux4b i878(.xtout(xtout[878]), .a(a[878]), .b(b[878]), .c(c[878]),
.d(d[878]), .sel(sel));
    xtmux4b i879(.xtout(xtout[879]), .a(a[879]), .b(b[879]), .c(c[879]),
.d(d[879]), .sel(sel));
    xtmux4b i880(.xtout(xtout[880]), .a(a[880]), .b(b[880]), .c(c[880]),
.d(d[880]), .sel(sel));
    xtmux4b i881(.xtout(xtout[881]), .a(a[881]), .b(b[881]), .c(c[881]),
.d(d[881]), .sel(sel));
```

```
    xtmux4b i882(.xtout(xtout[882]), .a(a[882]), .b(b[882]), .c(c[882]),
.d(d[882]), .sel(sel));
    xtmux4b i883(.xtout(xtout[883]), .a(a[883]), .b(b[883]), .c(c[883]),
.d(d[883]), .sel(sel));
    xtmux4b i884(.xtout(xtout[884]), .a(a[884]), .b(b[884]), .c(c[884]),
.d(d[884]), .sel(sel));
    xtmux4b i885(.xtout(xtout[885]), .a(a[885]), .b(b[885]), .c(c[885]),
.d(d[885]), .sel(sel));
    xtmux4b i886(.xtout(xtout[886]), .a(a[886]), .b(b[886]), .c(c[886]),
.d(d[886]), .sel(sel));
    xtmux4b i887(.xtout(xtout[887]), .a(a[887]), .b(b[887]), .c(c[887]),
.d(d[887]), .sel(sel));
    xtmux4b i888(.xtout(xtout[888]), .a(a[888]), .b(b[888]), .c(c[888]),
.d(d[888]), .sel(sel));
    xtmux4b i889(.xtout(xtout[889]), .a(a[889]), .b(b[889]), .c(c[889]),
.d(d[889]), .sel(sel));
    xtmux4b i890(.xtout(xtout[890]), .a(a[890]), .b(b[890]), .c(c[890]),
.d(d[890]), .sel(sel));
    xtmux4b i891(.xtout(xtout[891]), .a(a[891]), .b(b[891]), .c(c[891]),
.d(d[891]), .sel(sel));
    xtmux4b i892(.xtout(xtout[892]), .a(a[892]), .b(b[892]), .c(c[892]),
.d(d[892]), .sel(sel));
    xtmux4b i893(.xtout(xtout[893]), .a(a[893]), .b(b[893]), .c(c[893]),
.d(d[893]), .sel(sel));
    xtmux4b i894(.xtout(xtout[894]), .a(a[894]), .b(b[894]), .c(c[894]),
.d(d[894]), .sel(sel));
    xtmux4b i895(.xtout(xtout[895]), .a(a[895]), .b(b[895]), .c(c[895]),
.d(d[895]), .sel(sel));
    xtmux4b i896(.xtout(xtout[896]), .a(a[896]), .b(b[896]), .c(c[896]),
.d(d[896]), .sel(sel));
    xtmux4b i897(.xtout(xtout[897]), .a(a[897]), .b(b[897]), .c(c[897]),
.d(d[897]), .sel(sel));
    xtmux4b i898(.xtout(xtout[898]), .a(a[898]), .b(b[898]), .c(c[898]),
.d(d[898]), .sel(sel));
    xtmux4b i899(.xtout(xtout[899]), .a(a[899]), .b(b[899]), .c(c[899]),
.d(d[899]), .sel(sel));
    xtmux4b i900(.xtout(xtout[900]), .a(a[900]), .b(b[900]), .c(c[900]),
.d(d[900]), .sel(sel));
    xtmux4b i901(.xtout(xtout[901]), .a(a[901]), .b(b[901]), .c(c[901]),
.d(d[901]), .sel(sel));
    xtmux4b i902(.xtout(xtout[902]), .a(a[902]), .b(b[902]), .c(c[902]),
.d(d[902]), .sel(sel));
    xtmux4b i903(.xtout(xtout[903]), .a(a[903]), .b(b[903]), .c(c[903]),
.d(d[903]), .sel(sel));
    xtmux4b i904(.xtout(xtout[904]), .a(a[904]), .b(b[904]), .c(c[904]),
.d(d[904]), .sel(sel));
    xtmux4b i905(.xtout(xtout[905]), .a(a[905]), .b(b[905]), .c(c[905]),
.d(d[905]), .sel(sel));
    xtmux4b i906(.xtout(xtout[906]), .a(a[906]), .b(b[906]), .c(c[906]),
.d(d[906]), .sel(sel));
    xtmux4b i907(.xtout(xtout[907]), .a(a[907]), .b(b[907]), .c(c[907]),
.d(d[907]), .sel(sel));
    xtmux4b i908(.xtout(xtout[908]), .a(a[908]), .b(b[908]), .c(c[908]),
.d(d[908]), .sel(sel));
    xtmux4b i909(.xtout(xtout[909]), .a(a[909]), .b(b[909]), .c(c[909]),
.d(d[909]), .sel(sel));
```

```
    xtmux4b i910(.xtout(xtout[910]), .a(a[910]), .b(b[910]), .c(c[910]),
.d(d[910]), .sel(sel));
    xtmux4b i911(.xtout(xtout[911]), .a(a[911]), .b(b[911]), .c(c[911]),
.d(d[911]), .sel(sel));
    xtmux4b i912(.xtout(xtout[912]), .a(a[912]), .b(b[912]), .c(c[912]),
.d(d[912]), .sel(sel));
    xtmux4b i913(.xtout(xtout[913]), .a(a[913]), .b(b[913]), .c(c[913]),
.d(d[913]), .sel(sel));
    xtmux4b i914(.xtout(xtout[914]), .a(a[914]), .b(b[914]), .c(c[914]),
.d(d[914]), .sel(sel));
    xtmux4b i915(.xtout(xtout[915]), .a(a[915]), .b(b[915]), .c(c[915]),
.d(d[915]), .sel(sel));
    xtmux4b i916(.xtout(xtout[916]), .a(a[916]), .b(b[916]), .c(c[916]),
.d(d[916]), .sel(sel));
    xtmux4b i917(.xtout(xtout[917]), .a(a[917]), .b(b[917]), .c(c[917]),
.d(d[917]), .sel(sel));
    xtmux4b i918(.xtout(xtout[918]), .a(a[918]), .b(b[918]), .c(c[918]),
.d(d[918]), .sel(sel));
    xtmux4b i919(.xtout(xtout[919]), .a(a[919]), .b(b[919]), .c(c[919]),
.d(d[919]), .sel(sel));
    xtmux4b i920(.xtout(xtout[920]), .a(a[920]), .b(b[920]), .c(c[920]),
.d(d[920]), .sel(sel));
    xtmux4b i921(.xtout(xtout[921]), .a(a[921]), .b(b[921]), .c(c[921]),
.d(d[921]), .sel(sel));
    xtmux4b i922(.xtout(xtout[922]), .a(a[922]), .b(b[922]), .c(c[922]),
.d(d[922]), .sel(sel));
    xtmux4b i923(.xtout(xtout[923]), .a(a[923]), .b(b[923]), .c(c[923]),
.d(d[923]), .sel(sel));
    xtmux4b i924(.xtout(xtout[924]), .a(a[924]), .b(b[924]), .c(c[924]),
.d(d[924]), .sel(sel));
    xtmux4b i925(.xtout(xtout[925]), .a(a[925]), .b(b[925]), .c(c[925]),
.d(d[925]), .sel(sel));
    xtmux4b i926(.xtout(xtout[926]), .a(a[926]), .b(b[926]), .c(c[926]),
.d(d[926]), .sel(sel));
    xtmux4b i927(.xtout(xtout[927]), .a(a[927]), .b(b[927]), .c(c[927]),
.d(d[927]), .sel(sel));
    xtmux4b i928(.xtout(xtout[928]), .a(a[928]), .b(b[928]), .c(c[928]),
.d(d[928]), .sel(sel));
    xtmux4b i929(.xtout(xtout[929]), .a(a[929]), .b(b[929]), .c(c[929]),
.d(d[929]), .sel(sel));
    xtmux4b i930(.xtout(xtout[930]), .a(a[930]), .b(b[930]), .c(c[930]),
.d(d[930]), .sel(sel));
    xtmux4b i931(.xtout(xtout[931]), .a(a[931]), .b(b[931]), .c(c[931]),
.d(d[931]), .sel(sel));
    xtmux4b i932(.xtout(xtout[932]), .a(a[932]), .b(b[932]), .c(c[932]),
.d(d[932]), .sel(sel));
    xtmux4b i933(.xtout(xtout[933]), .a(a[933]), .b(b[933]), .c(c[933]),
.d(d[933]), .sel(sel));
    xtmux4b i934(.xtout(xtout[934]), .a(a[934]), .b(b[934]), .c(c[934]),
.d(d[934]), .sel(sel));
    xtmux4b i935(.xtout(xtout[935]), .a(a[935]), .b(b[935]), .c(c[935]),
.d(d[935]), .sel(sel));
    xtmux4b i936(.xtout(xtout[936]), .a(a[936]), .b(b[936]), .c(c[936]),
.d(d[936]), .sel(sel));
    xtmux4b i937(.xtout(xtout[937]), .a(a[937]), .b(b[937]), .c(c[937]),
.d(d[937]), .sel(sel));
```

```
    xtmux4b i938(.xtout(xtout[938]), .a(a[938]), .b(b[938]), .c(c[938]),
.d(d[938]), .sel(sel));
    xtmux4b i939(.xtout(xtout[939]), .a(a[939]), .b(b[939]), .c(c[939]),
.d(d[939]), .sel(sel));
    xtmux4b i940(.xtout(xtout[940]), .a(a[940]), .b(b[940]), .c(c[940]),
.d(d[940]), .sel(sel));
    xtmux4b i941(.xtout(xtout[941]), .a(a[941]), .b(b[941]), .c(c[941]),
.d(d[941]), .sel(sel));
    xtmux4b i942(.xtout(xtout[942]), .a(a[942]), .b(b[942]), .c(c[942]),
.d(d[942]), .sel(sel));
    xtmux4b i943(.xtout(xtout[943]), .a(a[943]), .b(b[943]), .c(c[943]),
.d(d[943]), .sel(sel));
    xtmux4b i944(.xtout(xtout[944]), .a(a[944]), .b(b[944]), .c(c[944]),
.d(d[944]), .sel(sel));
    xtmux4b i945(.xtout(xtout[945]), .a(a[945]), .b(b[945]), .c(c[945]),
.d(d[945]), .sel(sel));
    xtmux4b i946(.xtout(xtout[946]), .a(a[946]), .b(b[946]), .c(c[946]),
.d(d[946]), .sel(sel));
    xtmux4b i947(.xtout(xtout[947]), .a(a[947]), .b(b[947]), .c(c[947]),
.d(d[947]), .sel(sel));
    xtmux4b i948(.xtout(xtout[948]), .a(a[948]), .b(b[948]), .c(c[948]),
.d(d[948]), .sel(sel));
    xtmux4b i949(.xtout(xtout[949]), .a(a[949]), .b(b[949]), .c(c[949]),
.d(d[949]), .sel(sel));
    xtmux4b i950(.xtout(xtout[950]), .a(a[950]), .b(b[950]), .c(c[950]),
.d(d[950]), .sel(sel));
    xtmux4b i951(.xtout(xtout[951]), .a(a[951]), .b(b[951]), .c(c[951]),
.d(d[951]), .sel(sel));
    xtmux4b i952(.xtout(xtout[952]), .a(a[952]), .b(b[952]), .c(c[952]),
.d(d[952]), .sel(sel));
    xtmux4b i953(.xtout(xtout[953]), .a(a[953]), .b(b[953]), .c(c[953]),
.d(d[953]), .sel(sel));
    xtmux4b i954(.xtout(xtout[954]), .a(a[954]), .b(b[954]), .c(c[954]),
.d(d[954]), .sel(sel));
    xtmux4b i955(.xtout(xtout[955]), .a(a[955]), .b(b[955]), .c(c[955]),
.d(d[955]), .sel(sel));
    xtmux4b i956(.xtout(xtout[956]), .a(a[956]), .b(b[956]), .c(c[956]),
.d(d[956]), .sel(sel));
    xtmux4b i957(.xtout(xtout[957]), .a(a[957]), .b(b[957]), .c(c[957]),
.d(d[957]), .sel(sel));
    xtmux4b i958(.xtout(xtout[958]), .a(a[958]), .b(b[958]), .c(c[958]),
.d(d[958]), .sel(sel));
    xtmux4b i959(.xtout(xtout[959]), .a(a[959]), .b(b[959]), .c(c[959]),
.d(d[959]), .sel(sel));
    xtmux4b i960(.xtout(xtout[960]), .a(a[960]), .b(b[960]), .c(c[960]),
.d(d[960]), .sel(sel));
    xtmux4b i961(.xtout(xtout[961]), .a(a[961]), .b(b[961]), .c(c[961]),
.d(d[961]), .sel(sel));
    xtmux4b i962(.xtout(xtout[962]), .a(a[962]), .b(b[962]), .c(c[962]),
.d(d[962]), .sel(sel));
    xtmux4b i963(.xtout(xtout[963]), .a(a[963]), .b(b[963]), .c(c[963]),
.d(d[963]), .sel(sel));
    xtmux4b i964(.xtout(xtout[964]), .a(a[964]), .b(b[964]), .c(c[964]),
.d(d[964]), .sel(sel));
    xtmux4b i965(.xtout(xtout[965]), .a(a[965]), .b(b[965]), .c(c[965]),
.d(d[965]), .sel(sel));
```

```
    xtmux4b i966(.xtout(xtout[966]), .a(a[966]), .b(b[966]), .c(c[966]),
.d(d[966]), .sel(sel));
    xtmux4b i967(.xtout(xtout[967]), .a(a[967]), .b(b[967]), .c(c[967]),
.d(d[967]), .sel(sel));
    xtmux4b i968(.xtout(xtout[968]), .a(a[968]), .b(b[968]), .c(c[968]),
.d(d[968]), .sel(sel));
    xtmux4b i969(.xtout(xtout[969]), .a(a[969]), .b(b[969]), .c(c[969]),
.d(d[969]), .sel(sel));
    xtmux4b i970(.xtout(xtout[970]), .a(a[970]), .b(b[970]), .c(c[970]),
.d(d[970]), .sel(sel));
    xtmux4b i971(.xtout(xtout[971]), .a(a[971]), .b(b[971]), .c(c[971]),
.d(d[971]), .sel(sel));
    xtmux4b i972(.xtout(xtout[972]), .a(a[972]), .b(b[972]), .c(c[972]),
.d(d[972]), .sel(sel));
    xtmux4b i973(.xtout(xtout[973]), .a(a[973]), .b(b[973]), .c(c[973]),
.d(d[973]), .sel(sel));
    xtmux4b i974(.xtout(xtout[974]), .a(a[974]), .b(b[974]), .c(c[974]),
.d(d[974]), .sel(sel));
    xtmux4b i975(.xtout(xtout[975]), .a(a[975]), .b(b[975]), .c(c[975]),
.d(d[975]), .sel(sel));
    xtmux4b i976(.xtout(xtout[976]), .a(a[976]), .b(b[976]), .c(c[976]),
.d(d[976]), .sel(sel));
    xtmux4b i977(.xtout(xtout[977]), .a(a[977]), .b(b[977]), .c(c[977]),
.d(d[977]), .sel(sel));
    xtmux4b i978(.xtout(xtout[978]), .a(a[978]), .b(b[978]), .c(c[978]),
.d(d[978]), .sel(sel));
    xtmux4b i979(.xtout(xtout[979]), .a(a[979]), .b(b[979]), .c(c[979]),
.d(d[979]), .sel(sel));
    xtmux4b i980(.xtout(xtout[980]), .a(a[980]), .b(b[980]), .c(c[980]),
.d(d[980]), .sel(sel));
    xtmux4b i981(.xtout(xtout[981]), .a(a[981]), .b(b[981]), .c(c[981]),
.d(d[981]), .sel(sel));
    xtmux4b i982(.xtout(xtout[982]), .a(a[982]), .b(b[982]), .c(c[982]),
.d(d[982]), .sel(sel));
    xtmux4b i983(.xtout(xtout[983]), .a(a[983]), .b(b[983]), .c(c[983]),
.d(d[983]), .sel(sel));
    xtmux4b i984(.xtout(xtout[984]), .a(a[984]), .b(b[984]), .c(c[984]),
.d(d[984]), .sel(sel));
    xtmux4b i985(.xtout(xtout[985]), .a(a[985]), .b(b[985]), .c(c[985]),
.d(d[985]), .sel(sel));
    xtmux4b i986(.xtout(xtout[986]), .a(a[986]), .b(b[986]), .c(c[986]),
.d(d[986]), .sel(sel));
    xtmux4b i987(.xtout(xtout[987]), .a(a[987]), .b(b[987]), .c(c[987]),
.d(d[987]), .sel(sel));
    xtmux4b i988(.xtout(xtout[988]), .a(a[988]), .b(b[988]), .c(c[988]),
.d(d[988]), .sel(sel));
    xtmux4b i989(.xtout(xtout[989]), .a(a[989]), .b(b[989]), .c(c[989]),
.d(d[989]), .sel(sel));
    xtmux4b i990(.xtout(xtout[990]), .a(a[990]), .b(b[990]), .c(c[990]),
.d(d[990]), .sel(sel));
    xtmux4b i991(.xtout(xtout[991]), .a(a[991]), .b(b[991]), .c(c[991]),
.d(d[991]), .sel(sel));
    xtmux4b i992(.xtout(xtout[992]), .a(a[992]), .b(b[992]), .c(c[992]),
.d(d[992]), .sel(sel));
    xtmux4b i993(.xtout(xtout[993]), .a(a[993]), .b(b[993]), .c(c[993]),
.d(d[993]), .sel(sel));
```

```
    xtmux4b i994(.xtout(xtout[994]), .a(a[994]), .b(b[994]), .c(c[994]),
.d(d[994]), .sel(sel));
    xtmux4b i995(.xtout(xtout[995]), .a(a[995]), .b(b[995]), .c(c[995]),
.d(d[995]), .sel(sel));
    xtmux4b i996(.xtout(xtout[996]), .a(a[996]), .b(b[996]), .c(c[996]),
.d(d[996]), .sel(sel));
    xtmux4b i997(.xtout(xtout[997]), .a(a[997]), .b(b[997]), .c(c[997]),
.d(d[997]), .sel(sel));
    xtmux4b i998(.xtout(xtout[998]), .a(a[998]), .b(b[998]), .c(c[998]),
.d(d[998]), .sel(sel));
    xtmux4b i999(.xtout(xtout[999]), .a(a[999]), .b(b[999]), .c(c[999]),
.d(d[999]), .sel(sel));
    xtmux4b i1000(.xtout(xtout[1000]), .a(a[1000]), .b(b[1000]), .c(c[1000]),
.d(d[1000]), .sel(sel));
    xtmux4b i1001(.xtout(xtout[1001]), .a(a[1001]), .b(b[1001]), .c(c[1001]),
.d(d[1001]), .sel(sel));
    xtmux4b i1002(.xtout(xtout[1002]), .a(a[1002]), .b(b[1002]), .c(c[1002]),
.d(d[1002]), .sel(sel));
    xtmux4b i1003(.xtout(xtout[1003]), .a(a[1003]), .b(b[1003]), .c(c[1003]),
.d(d[1003]), .sel(sel));
    xtmux4b i1004(.xtout(xtout[1004]), .a(a[1004]), .b(b[1004]), .c(c[1004]),
.d(d[1004]), .sel(sel));
    xtmux4b i1005(.xtout(xtout[1005]), .a(a[1005]), .b(b[1005]), .c(c[1005]),
.d(d[1005]), .sel(sel));
    xtmux4b i1006(.xtout(xtout[1006]), .a(a[1006]), .b(b[1006]), .c(c[1006]),
.d(d[1006]), .sel(sel));
    xtmux4b i1007(.xtout(xtout[1007]), .a(a[1007]), .b(b[1007]), .c(c[1007]),
.d(d[1007]), .sel(sel));
    xtmux4b i1008(.xtout(xtout[1008]), .a(a[1008]), .b(b[1008]), .c(c[1008]),
.d(d[1008]), .sel(sel));
    xtmux4b i1009(.xtout(xtout[1009]), .a(a[1009]), .b(b[1009]), .c(c[1009]),
.d(d[1009]), .sel(sel));
    xtmux4b i1010(.xtout(xtout[1010]), .a(a[1010]), .b(b[1010]), .c(c[1010]),
.d(d[1010]), .sel(sel));
    xtmux4b i1011(.xtout(xtout[1011]), .a(a[1011]), .b(b[1011]), .c(c[1011]),
.d(d[1011]), .sel(sel));
    xtmux4b i1012(.xtout(xtout[1012]), .a(a[1012]), .b(b[1012]), .c(c[1012]),
.d(d[1012]), .sel(sel));
    xtmux4b i1013(.xtout(xtout[1013]), .a(a[1013]), .b(b[1013]), .c(c[1013]),
.d(d[1013]), .sel(sel));
    xtmux4b i1014(.xtout(xtout[1014]), .a(a[1014]), .b(b[1014]), .c(c[1014]),
.d(d[1014]), .sel(sel));
    xtmux4b i1015(.xtout(xtout[1015]), .a(a[1015]), .b(b[1015]), .c(c[1015]),
.d(d[1015]), .sel(sel));
    xtmux4b i1016(.xtout(xtout[1016]), .a(a[1016]), .b(b[1016]), .c(c[1016]),
.d(d[1016]), .sel(sel));
    xtmux4b i1017(.xtout(xtout[1017]), .a(a[1017]), .b(b[1017]), .c(c[1017]),
.d(d[1017]), .sel(sel));
    xtmux4b i1018(.xtout(xtout[1018]), .a(a[1018]), .b(b[1018]), .c(c[1018]),
.d(d[1018]), .sel(sel));
    xtmux4b i1019(.xtout(xtout[1019]), .a(a[1019]), .b(b[1019]), .c(c[1019]),
.d(d[1019]), .sel(sel));
    xtmux4b i1020(.xtout(xtout[1020]), .a(a[1020]), .b(b[1020]), .c(c[1020]),
.d(d[1020]), .sel(sel));
    xtmux4b i1021(.xtout(xtout[1021]), .a(a[1021]), .b(b[1021]), .c(c[1021]),
.d(d[1021]), .sel(sel));
```

```
    xtmux4b i1022(.xtout(xtout[1022]), .a(a[1022]), .b(b[1022]), .c(c[1022]),
.d(d[1022]), .sel(sel));
    xtmux4b i1023(.xtout(xtout[1023]), .a(a[1023]), .b(b[1023]), .c(c[1023]),
.d(d[1023]), .sel(sel));
endmodule
module xtcsa_1024(sum, carry, a, b, c);
output [1023:0] sum, carry;
input [1023:0] a, b, c;
    xtfa i0(.sum(sum[0]), .carry(carry[0]), .a(a[0]), .b(b[0]), .c(c[0]));
    xtfa i1(.sum(sum[1]), .carry(carry[1]), .a(a[1]), .b(b[1]), .c(c[1]));
    xtfa i2(.sum(sum[2]), .carry(carry[2]), .a(a[2]), .b(b[2]), .c(c[2]));
    xtfa i3(.sum(sum[3]), .carry(carry[3]), .a(a[3]), .b(b[3]), .c(c[3]));
    xtfa i4(.sum(sum[4]), .carry(carry[4]), .a(a[4]), .b(b[4]), .c(c[4]));
    xtfa i5(.sum(sum[5]), .carry(carry[5]), .a(a[5]), .b(b[5]), .c(c[5]));
    xtfa i6(.sum(sum[6]), .carry(carry[6]), .a(a[6]), .b(b[6]), .c(c[6]));
    xtfa i7(.sum(sum[7]), .carry(carry[7]), .a(a[7]), .b(b[7]), .c(c[7]));
    xtfa i8(.sum(sum[8]), .carry(carry[8]), .a(a[8]), .b(b[8]), .c(c[8]));
    xtfa i9(.sum(sum[9]), .carry(carry[9]), .a(a[9]), .b(b[9]), .c(c[9]));
    xtfa i10(.sum(sum[10]), .carry(carry[10]), .a(a[10]), .b(b[10]),
.c(c[10]));
    xtfa i11(.sum(sum[11]), .carry(carry[11]), .a(a[11]), .b(b[11]),
.c(c[11]));
    xtfa i12(.sum(sum[12]), .carry(carry[12]), .a(a[12]), .b(b[12]),
.c(c[12]));
    xtfa i13(.sum(sum[13]), .carry(carry[13]), .a(a[13]), .b(b[13]),
.c(c[13]));
    xtfa i14(.sum(sum[14]), .carry(carry[14]), .a(a[14]), .b(b[14]),
.c(c[14]));
    xtfa i15(.sum(sum[15]), .carry(carry[15]), .a(a[15]), .b(b[15]),
.c(c[15]));
    xtfa i16(.sum(sum[16]), .carry(carry[16]), .a(a[16]), .b(b[16]),
.c(c[16]));
    xtfa i17(.sum(sum[17]), .carry(carry[17]), .a(a[17]), .b(b[17]),
.c(c[17]));
    xtfa i18(.sum(sum[18]), .carry(carry[18]), .a(a[18]), .b(b[18]),
.c(c[18]));
    xtfa i19(.sum(sum[19]), .carry(carry[19]), .a(a[19]), .b(b[19]),
.c(c[19]));
    xtfa i20(.sum(sum[20]), .carry(carry[20]), .a(a[20]), .b(b[20]),
.c(c[20]));
    xtfa i21(.sum(sum[21]), .carry(carry[21]), .a(a[21]), .b(b[21]),
.c(c[21]));
    xtfa i22(.sum(sum[22]), .carry(carry[22]), .a(a[22]), .b(b[22]),
.c(c[22]));
    xtfa i23(.sum(sum[23]), .carry(carry[23]), .a(a[23]), .b(b[23]),
.c(c[23]));
    xtfa i24(.sum(sum[24]), .carry(carry[24]), .a(a[24]), .b(b[24]),
.c(c[24]));
    xtfa i25(.sum(sum[25]), .carry(carry[25]), .a(a[25]), .b(b[25]),
.c(c[25]));
    xtfa i26(.sum(sum[26]), .carry(carry[26]), .a(a[26]), .b(b[26]),
.c(c[26]));
    xtfa i27(.sum(sum[27]), .carry(carry[27]), .a(a[27]), .b(b[27]),
.c(c[27]));
    xtfa i28(.sum(sum[28]), .carry(carry[28]), .a(a[28]), .b(b[28]),
.c(c[28]));
```

```
    xtfa i29(.sum(sum[29]), .carry(carry[29]), .a(a[29]), .b(b[29]),
.c(c[29]));
    xtfa i30(.sum(sum[30]), .carry(carry[30]), .a(a[30]), .b(b[30]),
.c(c[30]));
    xtfa i31(.sum(sum[31]), .carry(carry[31]), .a(a[31]), .b(b[31]),
.c(c[31]));
    xtfa i32(.sum(sum[32]), .carry(carry[32]), .a(a[32]), .b(b[32]),
.c(c[32]));
    xtfa i33(.sum(sum[33]), .carry(carry[33]), .a(a[33]), .b(b[33]),
.c(c[33]));
    xtfa i34(.sum(sum[34]), .carry(carry[34]), .a(a[34]), .b(b[34]),
.c(c[34]));
    xtfa i35(.sum(sum[35]), .carry(carry[35]), .a(a[35]), .b(b[35]),
.c(c[35]));
    xtfa i36(.sum(sum[36]), .carry(carry[36]), .a(a[36]), .b(b[36]),
.c(c[36]));
    xtfa i37(.sum(sum[37]), .carry(carry[37]), .a(a[37]), .b(b[37]),
.c(c[37]));
    xtfa i38(.sum(sum[38]), .carry(carry[38]), .a(a[38]), .b(b[38]),
.c(c[38]));
    xtfa i39(.sum(sum[39]), .carry(carry[39]), .a(a[39]), .b(b[39]),
.c(c[39]));
    xtfa i40(.sum(sum[40]), .carry(carry[40]), .a(a[40]), .b(b[40]),
.c(c[40]));
    xtfa i41(.sum(sum[41]), .carry(carry[41]), .a(a[41]), .b(b[41]),
.c(c[41]));
    xtfa i42(.sum(sum[42]), .carry(carry[42]), .a(a[42]), .b(b[42]),
.c(c[42]));
    xtfa i43(.sum(sum[43]), .carry(carry[43]), .a(a[43]), .b(b[43]),
.c(c[43]));
    xtfa i44(.sum(sum[44]), .carry(carry[44]), .a(a[44]), .b(b[44]),
.c(c[44]));
    xtfa i45(.sum(sum[45]), .carry(carry[45]), .a(a[45]), .b(b[45]),
.c(c[45]));
    xtfa i46(.sum(sum[46]), .carry(carry[46]), .a(a[46]), .b(b[46]),
.c(c[46]));
    xtfa i47(.sum(sum[47]), .carry(carry[47]), .a(a[47]), .b(b[47]),
.c(c[47]));
    xtfa i48(.sum(sum[48]), .carry(carry[48]), .a(a[48]), .b(b[48]),
.c(c[48]));
    xtfa i49(.sum(sum[49]), .carry(carry[49]), .a(a[49]), .b(b[49]),
.c(c[49]));
    xtfa i50(.sum(sum[50]), .carry(carry[50]), .a(a[50]), .b(b[50]),
.c(c[50]));
    xtfa i51(.sum(sum[51]), .carry(carry[51]), .a(a[51]), .b(b[51]),
.c(c[51]));
    xtfa i52(.sum(sum[52]), .carry(carry[52]), .a(a[52]), .b(b[52]),
.c(c[52]));
    xtfa i53(.sum(sum[53]), .carry(carry[53]), .a(a[53]), .b(b[53]),
.c(c[53]));
    xtfa i54(.sum(sum[54]), .carry(carry[54]), .a(a[54]), .b(b[54]),
.c(c[54]));
    xtfa i55(.sum(sum[55]), .carry(carry[55]), .a(a[55]), .b(b[55]),
.c(c[55]));
    xtfa i56(.sum(sum[56]), .carry(carry[56]), .a(a[56]), .b(b[56]),
.c(c[56]));
```

```
    xtfa i57(.sum(sum[57]), .carry(carry[57]), .a(a[57]), .b(b[57]),
.c(c[57]));
    xtfa i58(.sum(sum[58]), .carry(carry[58]), .a(a[58]), .b(b[58]),
.c(c[58]));
    xtfa i59(.sum(sum[59]), .carry(carry[59]), .a(a[59]), .b(b[59]),
.c(c[59]));
    xtfa i60(.sum(sum[60]), .carry(carry[60]), .a(a[60]), .b(b[60]),
.c(c[60]));
    xtfa i61(.sum(sum[61]), .carry(carry[61]), .a(a[61]), .b(b[61]),
.c(c[61]));
    xtfa i62(.sum(sum[62]), .carry(carry[62]), .a(a[62]), .b(b[62]),
.c(c[62]));
    xtfa i63(.sum(sum[63]), .carry(carry[63]), .a(a[63]), .b(b[63]),
.c(c[63]));
    xtfa i64(.sum(sum[64]), .carry(carry[64]), .a(a[64]), .b(b[64]),
.c(c[64]));
    xtfa i65(.sum(sum[65]), .carry(carry[65]), .a(a[65]), .b(b[65]),
.c(c[65]));
    xtfa i66(.sum(sum[66]), .carry(carry[66]), .a(a[66]), .b(b[66]),
.c(c[66]));
    xtfa i67(.sum(sum[67]), .carry(carry[67]), .a(a[67]), .b(b[67]),
.c(c[67]));
    xtfa i68(.sum(sum[68]), .carry(carry[68]), .a(a[68]), .b(b[68]),
.c(c[68]));
    xtfa i69(.sum(sum[69]), .carry(carry[69]), .a(a[69]), .b(b[69]),
.c(c[69]));
    xtfa i70(.sum(sum[70]), .carry(carry[70]), .a(a[70]), .b(b[70]),
.c(c[70]));
    xtfa i71(.sum(sum[71]), .carry(carry[71]), .a(a[71]), .b(b[71]),
.c(c[71]));
    xtfa i72(.sum(sum[72]), .carry(carry[72]), .a(a[72]), .b(b[72]),
.c(c[72]));
    xtfa i73(.sum(sum[73]), .carry(carry[73]), .a(a[73]), .b(b[73]),
.c(c[73]));
    xtfa i74(.sum(sum[74]), .carry(carry[74]), .a(a[74]), .b(b[74]),
.c(c[74]));
    xtfa i75(.sum(sum[75]), .carry(carry[75]), .a(a[75]), .b(b[75]),
.c(c[75]));
    xtfa i76(.sum(sum[76]), .carry(carry[76]), .a(a[76]), .b(b[76]),
.c(c[76]));
    xtfa i77(.sum(sum[77]), .carry(carry[77]), .a(a[77]), .b(b[77]),
.c(c[77]));
    xtfa i78(.sum(sum[78]), .carry(carry[78]), .a(a[78]), .b(b[78]),
.c(c[78]));
    xtfa i79(.sum(sum[79]), .carry(carry[79]), .a(a[79]), .b(b[79]),
.c(c[79]));
    xtfa i80(.sum(sum[80]), .carry(carry[80]), .a(a[80]), .b(b[80]),
.c(c[80]));
    xtfa i81(.sum(sum[81]), .carry(carry[81]), .a(a[81]), .b(b[81]),
.c(c[81]));
    xtfa i82(.sum(sum[82]), .carry(carry[82]), .a(a[82]), .b(b[82]),
.c(c[82]));
    xtfa i83(.sum(sum[83]), .carry(carry[83]), .a(a[83]), .b(b[83]),
.c(c[83]));
    xtfa i84(.sum(sum[84]), .carry(carry[84]), .a(a[84]), .b(b[84]),
.c(c[84]));
```

```
    xtfa i85(.sum(sum[85]), .carry(carry[85]), .a(a[85]), .b(b[85]),
.c(c[85]));
    xtfa i86(.sum(sum[86]), .carry(carry[86]), .a(a[86]), .b(b[86]),
.c(c[86]));
    xtfa i87(.sum(sum[87]), .carry(carry[87]), .a(a[87]), .b(b[87]),
.c(c[87]));
    xtfa i88(.sum(sum[88]), .carry(carry[88]), .a(a[88]), .b(b[88]),
.c(c[88]));
    xtfa i89(.sum(sum[89]), .carry(carry[89]), .a(a[89]), .b(b[89]),
.c(c[89]));
    xtfa i90(.sum(sum[90]), .carry(carry[90]), .a(a[90]), .b(b[90]),
.c(c[90]));
    xtfa i91(.sum(sum[91]), .carry(carry[91]), .a(a[91]), .b(b[91]),
.c(c[91]));
    xtfa i92(.sum(sum[92]), .carry(carry[92]), .a(a[92]), .b(b[92]),
.c(c[92]));
    xtfa i93(.sum(sum[93]), .carry(carry[93]), .a(a[93]), .b(b[93]),
.c(c[93]));
    xtfa i94(.sum(sum[94]), .carry(carry[94]), .a(a[94]), .b(b[94]),
.c(c[94]));
    xtfa i95(.sum(sum[95]), .carry(carry[95]), .a(a[95]), .b(b[95]),
.c(c[95]));
    xtfa i96(.sum(sum[96]), .carry(carry[96]), .a(a[96]), .b(b[96]),
.c(c[96]));
    xtfa i97(.sum(sum[97]), .carry(carry[97]), .a(a[97]), .b(b[97]),
.c(c[97]));
    xtfa i98(.sum(sum[98]), .carry(carry[98]), .a(a[98]), .b(b[98]),
.c(c[98]));
    xtfa i99(.sum(sum[99]), .carry(carry[99]), .a(a[99]), .b(b[99]),
.c(c[99]));
    xtfa i100(.sum(sum[100]), .carry(carry[100]), .a(a[100]), .b(b[100]),
.c(c[100]));
    xtfa i101(.sum(sum[101]), .carry(carry[101]), .a(a[101]), .b(b[101]),
.c(c[101]));
    xtfa i102(.sum(sum[102]), .carry(carry[102]), .a(a[102]), .b(b[102]),
.c(c[102]));
    xtfa i103(.sum(sum[103]), .carry(carry[103]), .a(a[103]), .b(b[103]),
.c(c[103]));
    xtfa i104(.sum(sum[104]), .carry(carry[104]), .a(a[104]), .b(b[104]),
.c(c[104]));
    xtfa i105(.sum(sum[105]), .carry(carry[105]), .a(a[105]), .b(b[105]),
.c(c[105]));
    xtfa i106(.sum(sum[106]), .carry(carry[106]), .a(a[106]), .b(b[106]),
.c(c[106]));
    xtfa i107(.sum(sum[107]), .carry(carry[107]), .a(a[107]), .b(b[107]),
.c(c[107]));
    xtfa i108(.sum(sum[108]), .carry(carry[108]), .a(a[108]), .b(b[108]),
.c(c[108]));
    xtfa i109(.sum(sum[109]), .carry(carry[109]), .a(a[109]), .b(b[109]),
.c(c[109]));
    xtfa i110(.sum(sum[110]), .carry(carry[110]), .a(a[110]), .b(b[110]),
.c(c[110]));
    xtfa i111(.sum(sum[111]), .carry(carry[111]), .a(a[111]), .b(b[111]),
.c(c[111]));
    xtfa i112(.sum(sum[112]), .carry(carry[112]), .a(a[112]), .b(b[112]),
.c(c[112]));
```

```
    xtfa i113(.sum(sum[113]), .carry(carry[113]), .a(a[113]), .b(b[113]),
.c(c[113]));
    xtfa i114(.sum(sum[114]), .carry(carry[114]), .a(a[114]), .b(b[114]),
.c(c[114]));
    xtfa i115(.sum(sum[115]), .carry(carry[115]), .a(a[115]), .b(b[115]),
.c(c[115]));
    xtfa i116(.sum(sum[116]), .carry(carry[116]), .a(a[116]), .b(b[116]),
.c(c[116]));
    xtfa i117(.sum(sum[117]), .carry(carry[117]), .a(a[117]), .b(b[117]),
.c(c[117]));
    xtfa i118(.sum(sum[118]), .carry(carry[118]), .a(a[118]), .b(b[118]),
.c(c[118]));
    xtfa i119(.sum(sum[119]), .carry(carry[119]), .a(a[119]), .b(b[119]),
.c(c[119]));
    xtfa i120(.sum(sum[120]), .carry(carry[120]), .a(a[120]), .b(b[120]),
.c(c[120]));
    xtfa i121(.sum(sum[121]), .carry(carry[121]), .a(a[121]), .b(b[121]),
.c(c[121]));
    xtfa i122(.sum(sum[122]), .carry(carry[122]), .a(a[122]), .b(b[122]),
.c(c[122]));
    xtfa i123(.sum(sum[123]), .carry(carry[123]), .a(a[123]), .b(b[123]),
.c(c[123]));
    xtfa i124(.sum(sum[124]), .carry(carry[124]), .a(a[124]), .b(b[124]),
.c(c[124]));
    xtfa i125(.sum(sum[125]), .carry(carry[125]), .a(a[125]), .b(b[125]),
.c(c[125]));
    xtfa i126(.sum(sum[126]), .carry(carry[126]), .a(a[126]), .b(b[126]),
.c(c[126]));
    xtfa i127(.sum(sum[127]), .carry(carry[127]), .a(a[127]), .b(b[127]),
.c(c[127]));
    xtfa i128(.sum(sum[128]), .carry(carry[128]), .a(a[128]), .b(b[128]),
.c(c[128]));
    xtfa i129(.sum(sum[129]), .carry(carry[129]), .a(a[129]), .b(b[129]),
.c(c[129]));
    xtfa i130(.sum(sum[130]), .carry(carry[130]), .a(a[130]), .b(b[130]),
.c(c[130]));
    xtfa i131(.sum(sum[131]), .carry(carry[131]), .a(a[131]), .b(b[131]),
.c(c[131]));
    xtfa i132(.sum(sum[132]), .carry(carry[132]), .a(a[132]), .b(b[132]),
.c(c[132]));
    xtfa i133(.sum(sum[133]), .carry(carry[133]), .a(a[133]), .b(b[133]),
.c(c[133]));
    xtfa i134(.sum(sum[134]), .carry(carry[134]), .a(a[134]), .b(b[134]),
.c(c[134]));
    xtfa i135(.sum(sum[135]), .carry(carry[135]), .a(a[135]), .b(b[135]),
.c(c[135]));
    xtfa i136(.sum(sum[136]), .carry(carry[136]), .a(a[136]), .b(b[136]),
.c(c[136]));
    xtfa i137(.sum(sum[137]), .carry(carry[137]), .a(a[137]), .b(b[137]),
.c(c[137]));
    xtfa i138(.sum(sum[138]), .carry(carry[138]), .a(a[138]), .b(b[138]),
.c(c[138]));
    xtfa i139(.sum(sum[139]), .carry(carry[139]), .a(a[139]), .b(b[139]),
.c(c[139]));
    xtfa i140(.sum(sum[140]), .carry(carry[140]), .a(a[140]), .b(b[140]),
.c(c[140]));
```

```
    xtfa i141(.sum(sum[141]), .carry(carry[141]), .a(a[141]), .b(b[141]),
.c(c[141]));
    xtfa i142(.sum(sum[142]), .carry(carry[142]), .a(a[142]), .b(b[142]),
.c(c[142]));
    xtfa i143(.sum(sum[143]), .carry(carry[143]), .a(a[143]), .b(b[143]),
.c(c[143]));
    xtfa i144(.sum(sum[144]), .carry(carry[144]), .a(a[144]), .b(b[144]),
.c(c[144]));
    xtfa i145(.sum(sum[145]), .carry(carry[145]), .a(a[145]), .b(b[145]),
.c(c[145]));
    xtfa i146(.sum(sum[146]), .carry(carry[146]), .a(a[146]), .b(b[146]),
.c(c[146]));
    xtfa i147(.sum(sum[147]), .carry(carry[147]), .a(a[147]), .b(b[147]),
.c(c[147]));
    xtfa i148(.sum(sum[148]), .carry(carry[148]), .a(a[148]), .b(b[148]),
.c(c[148]));
    xtfa i149(.sum(sum[149]), .carry(carry[149]), .a(a[149]), .b(b[149]),
.c(c[149]));
    xtfa i150(.sum(sum[150]), .carry(carry[150]), .a(a[150]), .b(b[150]),
.c(c[150]));
    xtfa i151(.sum(sum[151]), .carry(carry[151]), .a(a[151]), .b(b[151]),
.c(c[151]));
    xtfa i152(.sum(sum[152]), .carry(carry[152]), .a(a[152]), .b(b[152]),
.c(c[152]));
    xtfa i153(.sum(sum[153]), .carry(carry[153]), .a(a[153]), .b(b[153]),
.c(c[153]));
    xtfa i154(.sum(sum[154]), .carry(carry[154]), .a(a[154]), .b(b[154]),
.c(c[154]));
    xtfa i155(.sum(sum[155]), .carry(carry[155]), .a(a[155]), .b(b[155]),
.c(c[155]));
    xtfa i156(.sum(sum[156]), .carry(carry[156]), .a(a[156]), .b(b[156]),
.c(c[156]));
    xtfa i157(.sum(sum[157]), .carry(carry[157]), .a(a[157]), .b(b[157]),
.c(c[157]));
    xtfa i158(.sum(sum[158]), .carry(carry[158]), .a(a[158]), .b(b[158]),
.c(c[158]));
    xtfa i159(.sum(sum[159]), .carry(carry[159]), .a(a[159]), .b(b[159]),
.c(c[159]));
    xtfa i160(.sum(sum[160]), .carry(carry[160]), .a(a[160]), .b(b[160]),
.c(c[160]));
    xtfa i161(.sum(sum[161]), .carry(carry[161]), .a(a[161]), .b(b[161]),
.c(c[161]));
    xtfa i162(.sum(sum[162]), .carry(carry[162]), .a(a[162]), .b(b[162]),
.c(c[162]));
    xtfa i163(.sum(sum[163]), .carry(carry[163]), .a(a[163]), .b(b[163]),
.c(c[163]));
    xtfa i164(.sum(sum[164]), .carry(carry[164]), .a(a[164]), .b(b[164]),
.c(c[164]));
    xtfa i165(.sum(sum[165]), .carry(carry[165]), .a(a[165]), .b(b[165]),
.c(c[165]));
    xtfa i166(.sum(sum[166]), .carry(carry[166]), .a(a[166]), .b(b[166]),
.c(c[166]));
    xtfa i167(.sum(sum[167]), .carry(carry[167]), .a(a[167]), .b(b[167]),
.c(c[167]));
    xtfa i168(.sum(sum[168]), .carry(carry[168]), .a(a[168]), .b(b[168]),
.c(c[168]));
```

```
    xtfa i169(.sum(sum[169]), .carry(carry[169]), .a(a[169]), .b(b[169]),
.c(c[169]));
    xtfa i170(.sum(sum[170]), .carry(carry[170]), .a(a[170]), .b(b[170]),
.c(c[170]));
    xtfa i171(.sum(sum[171]), .carry(carry[171]), .a(a[171]), .b(b[171]),
.c(c[171]));
    xtfa i172(.sum(sum[172]), .carry(carry[172]), .a(a[172]), .b(b[172]),
.c(c[172]));
    xtfa i173(.sum(sum[173]), .carry(carry[173]), .a(a[173]), .b(b[173]),
.c(c[173]));
    xtfa i174(.sum(sum[174]), .carry(carry[174]), .a(a[174]), .b(b[174]),
.c(c[174]));
    xtfa i175(.sum(sum[175]), .carry(carry[175]), .a(a[175]), .b(b[175]),
.c(c[175]));
    xtfa i176(.sum(sum[176]), .carry(carry[176]), .a(a[176]), .b(b[176]),
.c(c[176]));
    xtfa i177(.sum(sum[177]), .carry(carry[177]), .a(a[177]), .b(b[177]),
.c(c[177]));
    xtfa i178(.sum(sum[178]), .carry(carry[178]), .a(a[178]), .b(b[178]),
.c(c[178]));
    xtfa i179(.sum(sum[179]), .carry(carry[179]), .a(a[179]), .b(b[179]),
.c(c[179]));
    xtfa i180(.sum(sum[180]), .carry(carry[180]), .a(a[180]), .b(b[180]),
.c(c[180]));
    xtfa i181(.sum(sum[181]), .carry(carry[181]), .a(a[181]), .b(b[181]),
.c(c[181]));
    xtfa i182(.sum(sum[182]), .carry(carry[182]), .a(a[182]), .b(b[182]),
.c(c[182]));
    xtfa i183(.sum(sum[183]), .carry(carry[183]), .a(a[183]), .b(b[183]),
.c(c[183]));
    xtfa i184(.sum(sum[184]), .carry(carry[184]), .a(a[184]), .b(b[184]),
.c(c[184]));
    xtfa i185(.sum(sum[185]), .carry(carry[185]), .a(a[185]), .b(b[185]),
.c(c[185]));
    xtfa i186(.sum(sum[186]), .carry(carry[186]), .a(a[186]), .b(b[186]),
.c(c[186]));
    xtfa i187(.sum(sum[187]), .carry(carry[187]), .a(a[187]), .b(b[187]),
.c(c[187]));
    xtfa i188(.sum(sum[188]), .carry(carry[188]), .a(a[188]), .b(b[188]),
.c(c[188]));
    xtfa i189(.sum(sum[189]), .carry(carry[189]), .a(a[189]), .b(b[189]),
.c(c[189]));
    xtfa i190(.sum(sum[190]), .carry(carry[190]), .a(a[190]), .b(b[190]),
.c(c[190]));
    xtfa i191(.sum(sum[191]), .carry(carry[191]), .a(a[191]), .b(b[191]),
.c(c[191]));
    xtfa i192(.sum(sum[192]), .carry(carry[192]), .a(a[192]), .b(b[192]),
.c(c[192]));
    xtfa i193(.sum(sum[193]), .carry(carry[193]), .a(a[193]), .b(b[193]),
.c(c[193]));
    xtfa i194(.sum(sum[194]), .carry(carry[194]), .a(a[194]), .b(b[194]),
.c(c[194]));
    xtfa i195(.sum(sum[195]), .carry(carry[195]), .a(a[195]), .b(b[195]),
.c(c[195]));
    xtfa i196(.sum(sum[196]), .carry(carry[196]), .a(a[196]), .b(b[196]),
.c(c[196]));
```

```
    xtfa i197(.sum(sum[197]), .carry(carry[197]), .a(a[197]), .b(b[197]),
.c(c[197]));
    xtfa i198(.sum(sum[198]), .carry(carry[198]), .a(a[198]), .b(b[198]),
.c(c[198]));
    xtfa i199(.sum(sum[199]), .carry(carry[199]), .a(a[199]), .b(b[199]),
.c(c[199]));
    xtfa i200(.sum(sum[200]), .carry(carry[200]), .a(a[200]), .b(b[200]),
.c(c[200]));
    xtfa i201(.sum(sum[201]), .carry(carry[201]), .a(a[201]), .b(b[201]),
.c(c[201]));
    xtfa i202(.sum(sum[202]), .carry(carry[202]), .a(a[202]), .b(b[202]),
.c(c[202]));
    xtfa i203(.sum(sum[203]), .carry(carry[203]), .a(a[203]), .b(b[203]),
.c(c[203]));
    xtfa i204(.sum(sum[204]), .carry(carry[204]), .a(a[204]), .b(b[204]),
.c(c[204]));
    xtfa i205(.sum(sum[205]), .carry(carry[205]), .a(a[205]), .b(b[205]),
.c(c[205]));
    xtfa i206(.sum(sum[206]), .carry(carry[206]), .a(a[206]), .b(b[206]),
.c(c[206]));
    xtfa i207(.sum(sum[207]), .carry(carry[207]), .a(a[207]), .b(b[207]),
.c(c[207]));
    xtfa i208(.sum(sum[208]), .carry(carry[208]), .a(a[208]), .b(b[208]),
.c(c[208]));
    xtfa i209(.sum(sum[209]), .carry(carry[209]), .a(a[209]), .b(b[209]),
.c(c[209]));
    xtfa i210(.sum(sum[210]), .carry(carry[210]), .a(a[210]), .b(b[210]),
.c(c[210]));
    xtfa i211(.sum(sum[211]), .carry(carry[211]), .a(a[211]), .b(b[211]),
.c(c[211]));
    xtfa i212(.sum(sum[212]), .carry(carry[212]), .a(a[212]), .b(b[212]),
.c(c[212]));
    xtfa i213(.sum(sum[213]), .carry(carry[213]), .a(a[213]), .b(b[213]),
.c(c[213]));
    xtfa i214(.sum(sum[214]), .carry(carry[214]), .a(a[214]), .b(b[214]),
.c(c[214]));
    xtfa i215(.sum(sum[215]), .carry(carry[215]), .a(a[215]), .b(b[215]),
.c(c[215]));
    xtfa i216(.sum(sum[216]), .carry(carry[216]), .a(a[216]), .b(b[216]),
.c(c[216]));
    xtfa i217(.sum(sum[217]), .carry(carry[217]), .a(a[217]), .b(b[217]),
.c(c[217]));
    xtfa i218(.sum(sum[218]), .carry(carry[218]), .a(a[218]), .b(b[218]),
.c(c[218]));
    xtfa i219(.sum(sum[219]), .carry(carry[219]), .a(a[219]), .b(b[219]),
.c(c[219]));
    xtfa i220(.sum(sum[220]), .carry(carry[220]), .a(a[220]), .b(b[220]),
.c(c[220]));
    xtfa i221(.sum(sum[221]), .carry(carry[221]), .a(a[221]), .b(b[221]),
.c(c[221]));
    xtfa i222(.sum(sum[222]), .carry(carry[222]), .a(a[222]), .b(b[222]),
.c(c[222]));
    xtfa i223(.sum(sum[223]), .carry(carry[223]), .a(a[223]), .b(b[223]),
.c(c[223]));
    xtfa i224(.sum(sum[224]), .carry(carry[224]), .a(a[224]), .b(b[224]),
.c(c[224]));
```

```
    xtfa i225(.sum(sum[225]), .carry(carry[225]), .a(a[225]), .b(b[225]),
.c(c[225]));
    xtfa i226(.sum(sum[226]), .carry(carry[226]), .a(a[226]), .b(b[226]),
.c(c[226]));
    xtfa i227(.sum(sum[227]), .carry(carry[227]), .a(a[227]), .b(b[227]),
.c(c[227]));
    xtfa i228(.sum(sum[228]), .carry(carry[228]), .a(a[228]), .b(b[228]),
.c(c[228]));
    xtfa i229(.sum(sum[229]), .carry(carry[229]), .a(a[229]), .b(b[229]),
.c(c[229]));
    xtfa i230(.sum(sum[230]), .carry(carry[230]), .a(a[230]), .b(b[230]),
.c(c[230]));
    xtfa i231(.sum(sum[231]), .carry(carry[231]), .a(a[231]), .b(b[231]),
.c(c[231]));
    xtfa i232(.sum(sum[232]), .carry(carry[232]), .a(a[232]), .b(b[232]),
.c(c[232]));
    xtfa i233(.sum(sum[233]), .carry(carry[233]), .a(a[233]), .b(b[233]),
.c(c[233]));
    xtfa i234(.sum(sum[234]), .carry(carry[234]), .a(a[234]), .b(b[234]),
.c(c[234]));
    xtfa i235(.sum(sum[235]), .carry(carry[235]), .a(a[235]), .b(b[235]),
.c(c[235]));
    xtfa i236(.sum(sum[236]), .carry(carry[236]), .a(a[236]), .b(b[236]),
.c(c[236]));
    xtfa i237(.sum(sum[237]), .carry(carry[237]), .a(a[237]), .b(b[237]),
.c(c[237]));
    xtfa i238(.sum(sum[238]), .carry(carry[238]), .a(a[238]), .b(b[238]),
.c(c[238]));
    xtfa i239(.sum(sum[239]), .carry(carry[239]), .a(a[239]), .b(b[239]),
.c(c[239]));
    xtfa i240(.sum(sum[240]), .carry(carry[240]), .a(a[240]), .b(b[240]),
.c(c[240]));
    xtfa i241(.sum(sum[241]), .carry(carry[241]), .a(a[241]), .b(b[241]),
.c(c[241]));
    xtfa i242(.sum(sum[242]), .carry(carry[242]), .a(a[242]), .b(b[242]),
.c(c[242]));
    xtfa i243(.sum(sum[243]), .carry(carry[243]), .a(a[243]), .b(b[243]),
.c(c[243]));
    xtfa i244(.sum(sum[244]), .carry(carry[244]), .a(a[244]), .b(b[244]),
.c(c[244]));
    xtfa i245(.sum(sum[245]), .carry(carry[245]), .a(a[245]), .b(b[245]),
.c(c[245]));
    xtfa i246(.sum(sum[246]), .carry(carry[246]), .a(a[246]), .b(b[246]),
.c(c[246]));
    xtfa i247(.sum(sum[247]), .carry(carry[247]), .a(a[247]), .b(b[247]),
.c(c[247]));
    xtfa i248(.sum(sum[248]), .carry(carry[248]), .a(a[248]), .b(b[248]),
.c(c[248]));
    xtfa i249(.sum(sum[249]), .carry(carry[249]), .a(a[249]), .b(b[249]),
.c(c[249]));
    xtfa i250(.sum(sum[250]), .carry(carry[250]), .a(a[250]), .b(b[250]),
.c(c[250]));
    xtfa i251(.sum(sum[251]), .carry(carry[251]), .a(a[251]), .b(b[251]),
.c(c[251]));
    xtfa i252(.sum(sum[252]), .carry(carry[252]), .a(a[252]), .b(b[252]),
.c(c[252]));
```

```
    xtfa i253(.sum(sum[253]), .carry(carry[253]), .a(a[253]), .b(b[253]),
.c(c[253]));
    xtfa i254(.sum(sum[254]), .carry(carry[254]), .a(a[254]), .b(b[254]),
.c(c[254]));
    xtfa i255(.sum(sum[255]), .carry(carry[255]), .a(a[255]), .b(b[255]),
.c(c[255]));
    xtfa i256(.sum(sum[256]), .carry(carry[256]), .a(a[256]), .b(b[256]),
.c(c[256]));
    xtfa i257(.sum(sum[257]), .carry(carry[257]), .a(a[257]), .b(b[257]),
.c(c[257]));
    xtfa i258(.sum(sum[258]), .carry(carry[258]), .a(a[258]), .b(b[258]),
.c(c[258]));
    xtfa i259(.sum(sum[259]), .carry(carry[259]), .a(a[259]), .b(b[259]),
.c(c[259]));
    xtfa i260(.sum(sum[260]), .carry(carry[260]), .a(a[260]), .b(b[260]),
.c(c[260]));
    xtfa i261(.sum(sum[261]), .carry(carry[261]), .a(a[261]), .b(b[261]),
.c(c[261]));
    xtfa i262(.sum(sum[262]), .carry(carry[262]), .a(a[262]), .b(b[262]),
.c(c[262]));
    xtfa i263(.sum(sum[263]), .carry(carry[263]), .a(a[263]), .b(b[263]),
.c(c[263]));
    xtfa i264(.sum(sum[264]), .carry(carry[264]), .a(a[264]), .b(b[264]),
.c(c[264]));
    xtfa i265(.sum(sum[265]), .carry(carry[265]), .a(a[265]), .b(b[265]),
.c(c[265]));
    xtfa i266(.sum(sum[266]), .carry(carry[266]), .a(a[266]), .b(b[266]),
.c(c[266]));
    xtfa i267(.sum(sum[267]), .carry(carry[267]), .a(a[267]), .b(b[267]),
.c(c[267]));
    xtfa i268(.sum(sum[268]), .carry(carry[268]), .a(a[268]), .b(b[268]),
.c(c[268]));
    xtfa i269(.sum(sum[269]), .carry(carry[269]), .a(a[269]), .b(b[269]),
.c(c[269]));
    xtfa i270(.sum(sum[270]), .carry(carry[270]), .a(a[270]), .b(b[270]),
.c(c[270]));
    xtfa i271(.sum(sum[271]), .carry(carry[271]), .a(a[271]), .b(b[271]),
.c(c[271]));
    xtfa i272(.sum(sum[272]), .carry(carry[272]), .a(a[272]), .b(b[272]),
.c(c[272]));
    xtfa i273(.sum(sum[273]), .carry(carry[273]), .a(a[273]), .b(b[273]),
.c(c[273]));
    xtfa i274(.sum(sum[274]), .carry(carry[274]), .a(a[274]), .b(b[274]),
.c(c[274]));
    xtfa i275(.sum(sum[275]), .carry(carry[275]), .a(a[275]), .b(b[275]),
.c(c[275]));
    xtfa i276(.sum(sum[276]), .carry(carry[276]), .a(a[276]), .b(b[276]),
.c(c[276]));
    xtfa i277(.sum(sum[277]), .carry(carry[277]), .a(a[277]), .b(b[277]),
.c(c[277]));
    xtfa i278(.sum(sum[278]), .carry(carry[278]), .a(a[278]), .b(b[278]),
.c(c[278]));
    xtfa i279(.sum(sum[279]), .carry(carry[279]), .a(a[279]), .b(b[279]),
.c(c[279]));
    xtfa i280(.sum(sum[280]), .carry(carry[280]), .a(a[280]), .b(b[280]),
.c(c[280]));
```

```
    xtfa i281(.sum(sum[281]), .carry(carry[281]), .a(a[281]), .b(b[281]),
.c(c[281]));
    xtfa i282(.sum(sum[282]), .carry(carry[282]), .a(a[282]), .b(b[282]),
.c(c[282]));
    xtfa i283(.sum(sum[283]), .carry(carry[283]), .a(a[283]), .b(b[283]),
.c(c[283]));
    xtfa i284(.sum(sum[284]), .carry(carry[284]), .a(a[284]), .b(b[284]),
.c(c[284]));
    xtfa i285(.sum(sum[285]), .carry(carry[285]), .a(a[285]), .b(b[285]),
.c(c[285]));
    xtfa i286(.sum(sum[286]), .carry(carry[286]), .a(a[286]), .b(b[286]),
.c(c[286]));
    xtfa i287(.sum(sum[287]), .carry(carry[287]), .a(a[287]), .b(b[287]),
.c(c[287]));
    xtfa i288(.sum(sum[288]), .carry(carry[288]), .a(a[288]), .b(b[288]),
.c(c[288]));
    xtfa i289(.sum(sum[289]), .carry(carry[289]), .a(a[289]), .b(b[289]),
.c(c[289]));
    xtfa i290(.sum(sum[290]), .carry(carry[290]), .a(a[290]), .b(b[290]),
.c(c[290]));
    xtfa i291(.sum(sum[291]), .carry(carry[291]), .a(a[291]), .b(b[291]),
.c(c[291]));
    xtfa i292(.sum(sum[292]), .carry(carry[292]), .a(a[292]), .b(b[292]),
.c(c[292]));
    xtfa i293(.sum(sum[293]), .carry(carry[293]), .a(a[293]), .b(b[293]),
.c(c[293]));
    xtfa i294(.sum(sum[294]), .carry(carry[294]), .a(a[294]), .b(b[294]),
.c(c[294]));
    xtfa i295(.sum(sum[295]), .carry(carry[295]), .a(a[295]), .b(b[295]),
.c(c[295]));
    xtfa i296(.sum(sum[296]), .carry(carry[296]), .a(a[296]), .b(b[296]),
.c(c[296]));
    xtfa i297(.sum(sum[297]), .carry(carry[297]), .a(a[297]), .b(b[297]),
.c(c[297]));
    xtfa i298(.sum(sum[298]), .carry(carry[298]), .a(a[298]), .b(b[298]),
.c(c[298]));
    xtfa i299(.sum(sum[299]), .carry(carry[299]), .a(a[299]), .b(b[299]),
.c(c[299]));
    xtfa i300(.sum(sum[300]), .carry(carry[300]), .a(a[300]), .b(b[300]),
.c(c[300]));
    xtfa i301(.sum(sum[301]), .carry(carry[301]), .a(a[301]), .b(b[301]),
.c(c[301]));
    xtfa i302(.sum(sum[302]), .carry(carry[302]), .a(a[302]), .b(b[302]),
.c(c[302]));
    xtfa i303(.sum(sum[303]), .carry(carry[303]), .a(a[303]), .b(b[303]),
.c(c[303]));
    xtfa i304(.sum(sum[304]), .carry(carry[304]), .a(a[304]), .b(b[304]),
.c(c[304]));
    xtfa i305(.sum(sum[305]), .carry(carry[305]), .a(a[305]), .b(b[305]),
.c(c[305]));
    xtfa i306(.sum(sum[306]), .carry(carry[306]), .a(a[306]), .b(b[306]),
.c(c[306]));
    xtfa i307(.sum(sum[307]), .carry(carry[307]), .a(a[307]), .b(b[307]),
.c(c[307]));
    xtfa i308(.sum(sum[308]), .carry(carry[308]), .a(a[308]), .b(b[308]),
.c(c[308]));
```

```
    xtfa i309(.sum(sum[309]), .carry(carry[309]), .a(a[309]), .b(b[309]),
.c(c[309]));
    xtfa i310(.sum(sum[310]), .carry(carry[310]), .a(a[310]), .b(b[310]),
.c(c[310]));
    xtfa i311(.sum(sum[311]), .carry(carry[311]), .a(a[311]), .b(b[311]),
.c(c[311]));
    xtfa i312(.sum(sum[312]), .carry(carry[312]), .a(a[312]), .b(b[312]),
.c(c[312]));
    xtfa i313(.sum(sum[313]), .carry(carry[313]), .a(a[313]), .b(b[313]),
.c(c[313]));
    xtfa i314(.sum(sum[314]), .carry(carry[314]), .a(a[314]), .b(b[314]),
.c(c[314]));
    xtfa i315(.sum(sum[315]), .carry(carry[315]), .a(a[315]), .b(b[315]),
.c(c[315]));
    xtfa i316(.sum(sum[316]), .carry(carry[316]), .a(a[316]), .b(b[316]),
.c(c[316]));
    xtfa i317(.sum(sum[317]), .carry(carry[317]), .a(a[317]), .b(b[317]),
.c(c[317]));
    xtfa i318(.sum(sum[318]), .carry(carry[318]), .a(a[318]), .b(b[318]),
.c(c[318]));
    xtfa i319(.sum(sum[319]), .carry(carry[319]), .a(a[319]), .b(b[319]),
.c(c[319]));
    xtfa i320(.sum(sum[320]), .carry(carry[320]), .a(a[320]), .b(b[320]),
.c(c[320]));
    xtfa i321(.sum(sum[321]), .carry(carry[321]), .a(a[321]), .b(b[321]),
.c(c[321]));
    xtfa i322(.sum(sum[322]), .carry(carry[322]), .a(a[322]), .b(b[322]),
.c(c[322]));
    xtfa i323(.sum(sum[323]), .carry(carry[323]), .a(a[323]), .b(b[323]),
.c(c[323]));
    xtfa i324(.sum(sum[324]), .carry(carry[324]), .a(a[324]), .b(b[324]),
.c(c[324]));
    xtfa i325(.sum(sum[325]), .carry(carry[325]), .a(a[325]), .b(b[325]),
.c(c[325]));
    xtfa i326(.sum(sum[326]), .carry(carry[326]), .a(a[326]), .b(b[326]),
.c(c[326]));
    xtfa i327(.sum(sum[327]), .carry(carry[327]), .a(a[327]), .b(b[327]),
.c(c[327]));
    xtfa i328(.sum(sum[328]), .carry(carry[328]), .a(a[328]), .b(b[328]),
.c(c[328]));
    xtfa i329(.sum(sum[329]), .carry(carry[329]), .a(a[329]), .b(b[329]),
.c(c[329]));
    xtfa i330(.sum(sum[330]), .carry(carry[330]), .a(a[330]), .b(b[330]),
.c(c[330]));
    xtfa i331(.sum(sum[331]), .carry(carry[331]), .a(a[331]), .b(b[331]),
.c(c[331]));
    xtfa i332(.sum(sum[332]), .carry(carry[332]), .a(a[332]), .b(b[332]),
.c(c[332]));
    xtfa i333(.sum(sum[333]), .carry(carry[333]), .a(a[333]), .b(b[333]),
.c(c[333]));
    xtfa i334(.sum(sum[334]), .carry(carry[334]), .a(a[334]), .b(b[334]),
.c(c[334]));
    xtfa i335(.sum(sum[335]), .carry(carry[335]), .a(a[335]), .b(b[335]),
.c(c[335]));
    xtfa i336(.sum(sum[336]), .carry(carry[336]), .a(a[336]), .b(b[336]),
.c(c[336]));
```

```
    xtfa i337(.sum(sum[337]), .carry(carry[337]), .a(a[337]), .b(b[337]),
.c(c[337]));
    xtfa i338(.sum(sum[338]), .carry(carry[338]), .a(a[338]), .b(b[338]),
.c(c[338]));
    xtfa i339(.sum(sum[339]), .carry(carry[339]), .a(a[339]), .b(b[339]),
.c(c[339]));
    xtfa i340(.sum(sum[340]), .carry(carry[340]), .a(a[340]), .b(b[340]),
.c(c[340]));
    xtfa i341(.sum(sum[341]), .carry(carry[341]), .a(a[341]), .b(b[341]),
.c(c[341]));
    xtfa i342(.sum(sum[342]), .carry(carry[342]), .a(a[342]), .b(b[342]),
.c(c[342]));
    xtfa i343(.sum(sum[343]), .carry(carry[343]), .a(a[343]), .b(b[343]),
.c(c[343]));
    xtfa i344(.sum(sum[344]), .carry(carry[344]), .a(a[344]), .b(b[344]),
.c(c[344]));
    xtfa i345(.sum(sum[345]), .carry(carry[345]), .a(a[345]), .b(b[345]),
.c(c[345]));
    xtfa i346(.sum(sum[346]), .carry(carry[346]), .a(a[346]), .b(b[346]),
.c(c[346]));
    xtfa i347(.sum(sum[347]), .carry(carry[347]), .a(a[347]), .b(b[347]),
.c(c[347]));
    xtfa i348(.sum(sum[348]), .carry(carry[348]), .a(a[348]), .b(b[348]),
.c(c[348]));
    xtfa i349(.sum(sum[349]), .carry(carry[349]), .a(a[349]), .b(b[349]),
.c(c[349]));
    xtfa i350(.sum(sum[350]), .carry(carry[350]), .a(a[350]), .b(b[350]),
.c(c[350]));
    xtfa i351(.sum(sum[351]), .carry(carry[351]), .a(a[351]), .b(b[351]),
.c(c[351]));
    xtfa i352(.sum(sum[352]), .carry(carry[352]), .a(a[352]), .b(b[352]),
.c(c[352]));
    xtfa i353(.sum(sum[353]), .carry(carry[353]), .a(a[353]), .b(b[353]),
.c(c[353]));
    xtfa i354(.sum(sum[354]), .carry(carry[354]), .a(a[354]), .b(b[354]),
.c(c[354]));
    xtfa i355(.sum(sum[355]), .carry(carry[355]), .a(a[355]), .b(b[355]),
.c(c[355]));
    xtfa i356(.sum(sum[356]), .carry(carry[356]), .a(a[356]), .b(b[356]),
.c(c[356]));
    xtfa i357(.sum(sum[357]), .carry(carry[357]), .a(a[357]), .b(b[357]),
.c(c[357]));
    xtfa i358(.sum(sum[358]), .carry(carry[358]), .a(a[358]), .b(b[358]),
.c(c[358]));
    xtfa i359(.sum(sum[359]), .carry(carry[359]), .a(a[359]), .b(b[359]),
.c(c[359]));
    xtfa i360(.sum(sum[360]), .carry(carry[360]), .a(a[360]), .b(b[360]),
.c(c[360]));
    xtfa i361(.sum(sum[361]), .carry(carry[361]), .a(a[361]), .b(b[361]),
.c(c[361]));
    xtfa i362(.sum(sum[362]), .carry(carry[362]), .a(a[362]), .b(b[362]),
.c(c[362]));
    xtfa i363(.sum(sum[363]), .carry(carry[363]), .a(a[363]), .b(b[363]),
.c(c[363]));
    xtfa i364(.sum(sum[364]), .carry(carry[364]), .a(a[364]), .b(b[364]),
.c(c[364]));
```

```
    xtfa i365(.sum(sum[365]), .carry(carry[365]), .a(a[365]), .b(b[365]),
.c(c[365]));
    xtfa i366(.sum(sum[366]), .carry(carry[366]), .a(a[366]), .b(b[366]),
.c(c[366]));
    xtfa i367(.sum(sum[367]), .carry(carry[367]), .a(a[367]), .b(b[367]),
.c(c[367]));
    xtfa i368(.sum(sum[368]), .carry(carry[368]), .a(a[368]), .b(b[368]),
.c(c[368]));
    xtfa i369(.sum(sum[369]), .carry(carry[369]), .a(a[369]), .b(b[369]),
.c(c[369]));
    xtfa i370(.sum(sum[370]), .carry(carry[370]), .a(a[370]), .b(b[370]),
.c(c[370]));
    xtfa i371(.sum(sum[371]), .carry(carry[371]), .a(a[371]), .b(b[371]),
.c(c[371]));
    xtfa i372(.sum(sum[372]), .carry(carry[372]), .a(a[372]), .b(b[372]),
.c(c[372]));
    xtfa i373(.sum(sum[373]), .carry(carry[373]), .a(a[373]), .b(b[373]),
.c(c[373]));
    xtfa i374(.sum(sum[374]), .carry(carry[374]), .a(a[374]), .b(b[374]),
.c(c[374]));
    xtfa i375(.sum(sum[375]), .carry(carry[375]), .a(a[375]), .b(b[375]),
.c(c[375]));
    xtfa i376(.sum(sum[376]), .carry(carry[376]), .a(a[376]), .b(b[376]),
.c(c[376]));
    xtfa i377(.sum(sum[377]), .carry(carry[377]), .a(a[377]), .b(b[377]),
.c(c[377]));
    xtfa i378(.sum(sum[378]), .carry(carry[378]), .a(a[378]), .b(b[378]),
.c(c[378]));
    xtfa i379(.sum(sum[379]), .carry(carry[379]), .a(a[379]), .b(b[379]),
.c(c[379]));
    xtfa i380(.sum(sum[380]), .carry(carry[380]), .a(a[380]), .b(b[380]),
.c(c[380]));
    xtfa i381(.sum(sum[381]), .carry(carry[381]), .a(a[381]), .b(b[381]),
.c(c[381]));
    xtfa i382(.sum(sum[382]), .carry(carry[382]), .a(a[382]), .b(b[382]),
.c(c[382]));
    xtfa i383(.sum(sum[383]), .carry(carry[383]), .a(a[383]), .b(b[383]),
.c(c[383]));
    xtfa i384(.sum(sum[384]), .carry(carry[384]), .a(a[384]), .b(b[384]),
.c(c[384]));
    xtfa i385(.sum(sum[385]), .carry(carry[385]), .a(a[385]), .b(b[385]),
.c(c[385]));
    xtfa i386(.sum(sum[386]), .carry(carry[386]), .a(a[386]), .b(b[386]),
.c(c[386]));
    xtfa i387(.sum(sum[387]), .carry(carry[387]), .a(a[387]), .b(b[387]),
.c(c[387]));
    xtfa i388(.sum(sum[388]), .carry(carry[388]), .a(a[388]), .b(b[388]),
.c(c[388]));
    xtfa i389(.sum(sum[389]), .carry(carry[389]), .a(a[389]), .b(b[389]),
.c(c[389]));
    xtfa i390(.sum(sum[390]), .carry(carry[390]), .a(a[390]), .b(b[390]),
.c(c[390]));
    xtfa i391(.sum(sum[391]), .carry(carry[391]), .a(a[391]), .b(b[391]),
.c(c[391]));
    xtfa i392(.sum(sum[392]), .carry(carry[392]), .a(a[392]), .b(b[392]),
.c(c[392]));
```

```
    xtfa i393(.sum(sum[393]), .carry(carry[393]), .a(a[393]), .b(b[393]),
.c(c[393]));
    xtfa i394(.sum(sum[394]), .carry(carry[394]), .a(a[394]), .b(b[394]),
.c(c[394]));
    xtfa i395(.sum(sum[395]), .carry(carry[395]), .a(a[395]), .b(b[395]),
.c(c[395]));
    xtfa i396(.sum(sum[396]), .carry(carry[396]), .a(a[396]), .b(b[396]),
.c(c[396]));
    xtfa i397(.sum(sum[397]), .carry(carry[397]), .a(a[397]), .b(b[397]),
.c(c[397]));
    xtfa i398(.sum(sum[398]), .carry(carry[398]), .a(a[398]), .b(b[398]),
.c(c[398]));
    xtfa i399(.sum(sum[399]), .carry(carry[399]), .a(a[399]), .b(b[399]),
.c(c[399]));
    xtfa i400(.sum(sum[400]), .carry(carry[400]), .a(a[400]), .b(b[400]),
.c(c[400]));
    xtfa i401(.sum(sum[401]), .carry(carry[401]), .a(a[401]), .b(b[401]),
.c(c[401]));
    xtfa i402(.sum(sum[402]), .carry(carry[402]), .a(a[402]), .b(b[402]),
.c(c[402]));
    xtfa i403(.sum(sum[403]), .carry(carry[403]), .a(a[403]), .b(b[403]),
.c(c[403]));
    xtfa i404(.sum(sum[404]), .carry(carry[404]), .a(a[404]), .b(b[404]),
.c(c[404]));
    xtfa i405(.sum(sum[405]), .carry(carry[405]), .a(a[405]), .b(b[405]),
.c(c[405]));
    xtfa i406(.sum(sum[406]), .carry(carry[406]), .a(a[406]), .b(b[406]),
.c(c[406]));
    xtfa i407(.sum(sum[407]), .carry(carry[407]), .a(a[407]), .b(b[407]),
.c(c[407]));
    xtfa i408(.sum(sum[408]), .carry(carry[408]), .a(a[408]), .b(b[408]),
.c(c[408]));
    xtfa i409(.sum(sum[409]), .carry(carry[409]), .a(a[409]), .b(b[409]),
.c(c[409]));
    xtfa i410(.sum(sum[410]), .carry(carry[410]), .a(a[410]), .b(b[410]),
.c(c[410]));
    xtfa i411(.sum(sum[411]), .carry(carry[411]), .a(a[411]), .b(b[411]),
.c(c[411]));
    xtfa i412(.sum(sum[412]), .carry(carry[412]), .a(a[412]), .b(b[412]),
.c(c[412]));
    xtfa i413(.sum(sum[413]), .carry(carry[413]), .a(a[413]), .b(b[413]),
.c(c[413]));
    xtfa i414(.sum(sum[414]), .carry(carry[414]), .a(a[414]), .b(b[414]),
.c(c[414]));
    xtfa i415(.sum(sum[415]), .carry(carry[415]), .a(a[415]), .b(b[415]),
.c(c[415]));
    xtfa i416(.sum(sum[416]), .carry(carry[416]), .a(a[416]), .b(b[416]),
.c(c[416]));
    xtfa i417(.sum(sum[417]), .carry(carry[417]), .a(a[417]), .b(b[417]),
.c(c[417]));
    xtfa i418(.sum(sum[418]), .carry(carry[418]), .a(a[418]), .b(b[418]),
.c(c[418]));
    xtfa i419(.sum(sum[419]), .carry(carry[419]), .a(a[419]), .b(b[419]),
.c(c[419]));
    xtfa i420(.sum(sum[420]), .carry(carry[420]), .a(a[420]), .b(b[420]),
.c(c[420]));
```

```
     xtfa i421(.sum(sum[421]), .carry(carry[421]), .a(a[421]), .b(b[421]),
 .c(c[421]));
     xtfa i422(.sum(sum[422]), .carry(carry[422]), .a(a[422]), .b(b[422]),
 .c(c[422]));
     xtfa i423(.sum(sum[423]), .carry(carry[423]), .a(a[423]), .b(b[423]),
 .c(c[423]));
     xtfa i424(.sum(sum[424]), .carry(carry[424]), .a(a[424]), .b(b[424]),
 .c(c[424]));
     xtfa i425(.sum(sum[425]), .carry(carry[425]), .a(a[425]), .b(b[425]),
 .c(c[425]));
     xtfa i426(.sum(sum[426]), .carry(carry[426]), .a(a[426]), .b(b[426]),
 .c(c[426]));
     xtfa i427(.sum(sum[427]), .carry(carry[427]), .a(a[427]), .b(b[427]),
 .c(c[427]));
     xtfa i428(.sum(sum[428]), .carry(carry[428]), .a(a[428]), .b(b[428]),
 .c(c[428]));
     xtfa i429(.sum(sum[429]), .carry(carry[429]), .a(a[429]), .b(b[429]),
 .c(c[429]));
     xtfa i430(.sum(sum[430]), .carry(carry[430]), .a(a[430]), .b(b[430]),
 .c(c[430]));
     xtfa i431(.sum(sum[431]), .carry(carry[431]), .a(a[431]), .b(b[431]),
 .c(c[431]));
     xtfa i432(.sum(sum[432]), .carry(carry[432]), .a(a[432]), .b(b[432]),
 .c(c[432]));
     xtfa i433(.sum(sum[433]), .carry(carry[433]), .a(a[433]), .b(b[433]),
 .c(c[433]));
     xtfa i434(.sum(sum[434]), .carry(carry[434]), .a(a[434]), .b(b[434]),
 .c(c[434]));
     xtfa i435(.sum(sum[435]), .carry(carry[435]), .a(a[435]), .b(b[435]),
 .c(c[435]));
     xtfa i436(.sum(sum[436]), .carry(carry[436]), .a(a[436]), .b(b[436]),
 .c(c[436]));
     xtfa i437(.sum(sum[437]), .carry(carry[437]), .a(a[437]), .b(b[437]),
 .c(c[437]));
     xtfa i438(.sum(sum[438]), .carry(carry[438]), .a(a[438]), .b(b[438]),
 .c(c[438]));
     xtfa i439(.sum(sum[439]), .carry(carry[439]), .a(a[439]), .b(b[439]),
 .c(c[439]));
     xtfa i440(.sum(sum[440]), .carry(carry[440]), .a(a[440]), .b(b[440]),
 .c(c[440]));
     xtfa i441(.sum(sum[441]), .carry(carry[441]), .a(a[441]), .b(b[441]),
 .c(c[441]));
     xtfa i442(.sum(sum[442]), .carry(carry[442]), .a(a[442]), .b(b[442]),
 .c(c[442]));
     xtfa i443(.sum(sum[443]), .carry(carry[443]), .a(a[443]), .b(b[443]),
 .c(c[443]));
     xtfa i444(.sum(sum[444]), .carry(carry[444]), .a(a[444]), .b(b[444]),
 .c(c[444]));
     xtfa i445(.sum(sum[445]), .carry(carry[445]), .a(a[445]), .b(b[445]),
 .c(c[445]));
     xtfa i446(.sum(sum[446]), .carry(carry[446]), .a(a[446]), .b(b[446]),
 .c(c[446]));
     xtfa i447(.sum(sum[447]), .carry(carry[447]), .a(a[447]), .b(b[447]),
 .c(c[447]));
     xtfa i448(.sum(sum[448]), .carry(carry[448]), .a(a[448]), .b(b[448]),
 .c(c[448]));
```

```
    xtfa i449(.sum(sum[449]), .carry(carry[449]), .a(a[449]), .b(b[449]),
.c(c[449]));
    xtfa i450(.sum(sum[450]), .carry(carry[450]), .a(a[450]), .b(b[450]),
.c(c[450]));
    xtfa i451(.sum(sum[451]), .carry(carry[451]), .a(a[451]), .b(b[451]),
.c(c[451]));
    xtfa i452(.sum(sum[452]), .carry(carry[452]), .a(a[452]), .b(b[452]),
.c(c[452]));
    xtfa i453(.sum(sum[453]), .carry(carry[453]), .a(a[453]), .b(b[453]),
.c(c[453]));
    xtfa i454(.sum(sum[454]), .carry(carry[454]), .a(a[454]), .b(b[454]),
.c(c[454]));
    xtfa i455(.sum(sum[455]), .carry(carry[455]), .a(a[455]), .b(b[455]),
.c(c[455]));
    xtfa i456(.sum(sum[456]), .carry(carry[456]), .a(a[456]), .b(b[456]),
.c(c[456]));
    xtfa i457(.sum(sum[457]), .carry(carry[457]), .a(a[457]), .b(b[457]),
.c(c[457]));
    xtfa i458(.sum(sum[458]), .carry(carry[458]), .a(a[458]), .b(b[458]),
.c(c[458]));
    xtfa i459(.sum(sum[459]), .carry(carry[459]), .a(a[459]), .b(b[459]),
.c(c[459]));
    xtfa i460(.sum(sum[460]), .carry(carry[460]), .a(a[460]), .b(b[460]),
.c(c[460]));
    xtfa i461(.sum(sum[461]), .carry(carry[461]), .a(a[461]), .b(b[461]),
.c(c[461]));
    xtfa i462(.sum(sum[462]), .carry(carry[462]), .a(a[462]), .b(b[462]),
.c(c[462]));
    xtfa i463(.sum(sum[463]), .carry(carry[463]), .a(a[463]), .b(b[463]),
.c(c[463]));
    xtfa i464(.sum(sum[464]), .carry(carry[464]), .a(a[464]), .b(b[464]),
.c(c[464]));
    xtfa i465(.sum(sum[465]), .carry(carry[465]), .a(a[465]), .b(b[465]),
.c(c[465]));
    xtfa i466(.sum(sum[466]), .carry(carry[466]), .a(a[466]), .b(b[466]),
.c(c[466]));
    xtfa i467(.sum(sum[467]), .carry(carry[467]), .a(a[467]), .b(b[467]),
.c(c[467]));
    xtfa i468(.sum(sum[468]), .carry(carry[468]), .a(a[468]), .b(b[468]),
.c(c[468]));
    xtfa i469(.sum(sum[469]), .carry(carry[469]), .a(a[469]), .b(b[469]),
.c(c[469]));
    xtfa i470(.sum(sum[470]), .carry(carry[470]), .a(a[470]), .b(b[470]),
.c(c[470]));
    xtfa i471(.sum(sum[471]), .carry(carry[471]), .a(a[471]), .b(b[471]),
.c(c[471]));
    xtfa i472(.sum(sum[472]), .carry(carry[472]), .a(a[472]), .b(b[472]),
.c(c[472]));
    xtfa i473(.sum(sum[473]), .carry(carry[473]), .a(a[473]), .b(b[473]),
.c(c[473]));
    xtfa i474(.sum(sum[474]), .carry(carry[474]), .a(a[474]), .b(b[474]),
.c(c[474]));
    xtfa i475(.sum(sum[475]), .carry(carry[475]), .a(a[475]), .b(b[475]),
.c(c[475]));
    xtfa i476(.sum(sum[476]), .carry(carry[476]), .a(a[476]), .b(b[476]),
.c(c[476]));
```

```
    xtfa i477(.sum(sum[477]), .carry(carry[477]), .a(a[477]), .b(b[477]),
.c(c[477]));
    xtfa i478(.sum(sum[478]), .carry(carry[478]), .a(a[478]), .b(b[478]),
.c(c[478]));
    xtfa i479(.sum(sum[479]), .carry(carry[479]), .a(a[479]), .b(b[479]),
.c(c[479]));
    xtfa i480(.sum(sum[480]), .carry(carry[480]), .a(a[480]), .b(b[480]),
.c(c[480]));
    xtfa i481(.sum(sum[481]), .carry(carry[481]), .a(a[481]), .b(b[481]),
.c(c[481]));
    xtfa i482(.sum(sum[482]), .carry(carry[482]), .a(a[482]), .b(b[482]),
.c(c[482]));
    xtfa i483(.sum(sum[483]), .carry(carry[483]), .a(a[483]), .b(b[483]),
.c(c[483]));
    xtfa i484(.sum(sum[484]), .carry(carry[484]), .a(a[484]), .b(b[484]),
.c(c[484]));
    xtfa i485(.sum(sum[485]), .carry(carry[485]), .a(a[485]), .b(b[485]),
.c(c[485]));
    xtfa i486(.sum(sum[486]), .carry(carry[486]), .a(a[486]), .b(b[486]),
.c(c[486]));
    xtfa i487(.sum(sum[487]), .carry(carry[487]), .a(a[487]), .b(b[487]),
.c(c[487]));
    xtfa i488(.sum(sum[488]), .carry(carry[488]), .a(a[488]), .b(b[488]),
.c(c[488]));
    xtfa i489(.sum(sum[489]), .carry(carry[489]), .a(a[489]), .b(b[489]),
.c(c[489]));
    xtfa i490(.sum(sum[490]), .carry(carry[490]), .a(a[490]), .b(b[490]),
.c(c[490]));
    xtfa i491(.sum(sum[491]), .carry(carry[491]), .a(a[491]), .b(b[491]),
.c(c[491]));
    xtfa i492(.sum(sum[492]), .carry(carry[492]), .a(a[492]), .b(b[492]),
.c(c[492]));
    xtfa i493(.sum(sum[493]), .carry(carry[493]), .a(a[493]), .b(b[493]),
.c(c[493]));
    xtfa i494(.sum(sum[494]), .carry(carry[494]), .a(a[494]), .b(b[494]),
.c(c[494]));
    xtfa i495(.sum(sum[495]), .carry(carry[495]), .a(a[495]), .b(b[495]),
.c(c[495]));
    xtfa i496(.sum(sum[496]), .carry(carry[496]), .a(a[496]), .b(b[496]),
.c(c[496]));
    xtfa i497(.sum(sum[497]), .carry(carry[497]), .a(a[497]), .b(b[497]),
.c(c[497]));
    xtfa i498(.sum(sum[498]), .carry(carry[498]), .a(a[498]), .b(b[498]),
.c(c[498]));
    xtfa i499(.sum(sum[499]), .carry(carry[499]), .a(a[499]), .b(b[499]),
.c(c[499]));
    xtfa i500(.sum(sum[500]), .carry(carry[500]), .a(a[500]), .b(b[500]),
.c(c[500]));
    xtfa i501(.sum(sum[501]), .carry(carry[501]), .a(a[501]), .b(b[501]),
.c(c[501]));
    xtfa i502(.sum(sum[502]), .carry(carry[502]), .a(a[502]), .b(b[502]),
.c(c[502]));
    xtfa i503(.sum(sum[503]), .carry(carry[503]), .a(a[503]), .b(b[503]),
.c(c[503]));
    xtfa i504(.sum(sum[504]), .carry(carry[504]), .a(a[504]), .b(b[504]),
.c(c[504]));
```

```
    xtfa i505(.sum(sum[505]), .carry(carry[505]), .a(a[505]), .b(b[505]),
.c(c[505]));
    xtfa i506(.sum(sum[506]), .carry(carry[506]), .a(a[506]), .b(b[506]),
.c(c[506]));
    xtfa i507(.sum(sum[507]), .carry(carry[507]), .a(a[507]), .b(b[507]),
.c(c[507]));
    xtfa i508(.sum(sum[508]), .carry(carry[508]), .a(a[508]), .b(b[508]),
.c(c[508]));
    xtfa i509(.sum(sum[509]), .carry(carry[509]), .a(a[509]), .b(b[509]),
.c(c[509]));
    xtfa i510(.sum(sum[510]), .carry(carry[510]), .a(a[510]), .b(b[510]),
.c(c[510]));
    xtfa i511(.sum(sum[511]), .carry(carry[511]), .a(a[511]), .b(b[511]),
.c(c[511]));
    xtfa i512(.sum(sum[512]), .carry(carry[512]), .a(a[512]), .b(b[512]),
.c(c[512]));
    xtfa i513(.sum(sum[513]), .carry(carry[513]), .a(a[513]), .b(b[513]),
.c(c[513]));
    xtfa i514(.sum(sum[514]), .carry(carry[514]), .a(a[514]), .b(b[514]),
.c(c[514]));
    xtfa i515(.sum(sum[515]), .carry(carry[515]), .a(a[515]), .b(b[515]),
.c(c[515]));
    xtfa i516(.sum(sum[516]), .carry(carry[516]), .a(a[516]), .b(b[516]),
.c(c[516]));
    xtfa i517(.sum(sum[517]), .carry(carry[517]), .a(a[517]), .b(b[517]),
.c(c[517]));
    xtfa i518(.sum(sum[518]), .carry(carry[518]), .a(a[518]), .b(b[518]),
.c(c[518]));
    xtfa i519(.sum(sum[519]), .carry(carry[519]), .a(a[519]), .b(b[519]),
.c(c[519]));
    xtfa i520(.sum(sum[520]), .carry(carry[520]), .a(a[520]), .b(b[520]),
.c(c[520]));
    xtfa i521(.sum(sum[521]), .carry(carry[521]), .a(a[521]), .b(b[521]),
.c(c[521]));
    xtfa i522(.sum(sum[522]), .carry(carry[522]), .a(a[522]), .b(b[522]),
.c(c[522]));
    xtfa i523(.sum(sum[523]), .carry(carry[523]), .a(a[523]), .b(b[523]),
.c(c[523]));
    xtfa i524(.sum(sum[524]), .carry(carry[524]), .a(a[524]), .b(b[524]),
.c(c[524]));
    xtfa i525(.sum(sum[525]), .carry(carry[525]), .a(a[525]), .b(b[525]),
.c(c[525]));
    xtfa i526(.sum(sum[526]), .carry(carry[526]), .a(a[526]), .b(b[526]),
.c(c[526]));
    xtfa i527(.sum(sum[527]), .carry(carry[527]), .a(a[527]), .b(b[527]),
.c(c[527]));
    xtfa i528(.sum(sum[528]), .carry(carry[528]), .a(a[528]), .b(b[528]),
.c(c[528]));
    xtfa i529(.sum(sum[529]), .carry(carry[529]), .a(a[529]), .b(b[529]),
.c(c[529]));
    xtfa i530(.sum(sum[530]), .carry(carry[530]), .a(a[530]), .b(b[530]),
.c(c[530]));
    xtfa i531(.sum(sum[531]), .carry(carry[531]), .a(a[531]), .b(b[531]),
.c(c[531]));
    xtfa i532(.sum(sum[532]), .carry(carry[532]), .a(a[532]), .b(b[532]),
.c(c[532]));
```

```
    xtfa i533(.sum(sum[533]), .carry(carry[533]), .a(a[533]), .b(b[533]),
.c(c[533]));
    xtfa i534(.sum(sum[534]), .carry(carry[534]), .a(a[534]), .b(b[534]),
.c(c[534]));
    xtfa i535(.sum(sum[535]), .carry(carry[535]), .a(a[535]), .b(b[535]),
.c(c[535]));
    xtfa i536(.sum(sum[536]), .carry(carry[536]), .a(a[536]), .b(b[536]),
.c(c[536]));
    xtfa i537(.sum(sum[537]), .carry(carry[537]), .a(a[537]), .b(b[537]),
.c(c[537]));
    xtfa i538(.sum(sum[538]), .carry(carry[538]), .a(a[538]), .b(b[538]),
.c(c[538]));
    xtfa i539(.sum(sum[539]), .carry(carry[539]), .a(a[539]), .b(b[539]),
.c(c[539]));
    xtfa i540(.sum(sum[540]), .carry(carry[540]), .a(a[540]), .b(b[540]),
.c(c[540]));
    xtfa i541(.sum(sum[541]), .carry(carry[541]), .a(a[541]), .b(b[541]),
.c(c[541]));
    xtfa i542(.sum(sum[542]), .carry(carry[542]), .a(a[542]), .b(b[542]),
.c(c[542]));
    xtfa i543(.sum(sum[543]), .carry(carry[543]), .a(a[543]), .b(b[543]),
.c(c[543]));
    xtfa i544(.sum(sum[544]), .carry(carry[544]), .a(a[544]), .b(b[544]),
.c(c[544]));
    xtfa i545(.sum(sum[545]), .carry(carry[545]), .a(a[545]), .b(b[545]),
.c(c[545]));
    xtfa i546(.sum(sum[546]), .carry(carry[546]), .a(a[546]), .b(b[546]),
.c(c[546]));
    xtfa i547(.sum(sum[547]), .carry(carry[547]), .a(a[547]), .b(b[547]),
.c(c[547]));
    xtfa i548(.sum(sum[548]), .carry(carry[548]), .a(a[548]), .b(b[548]),
.c(c[548]));
    xtfa i549(.sum(sum[549]), .carry(carry[549]), .a(a[549]), .b(b[549]),
.c(c[549]));
    xtfa i550(.sum(sum[550]), .carry(carry[550]), .a(a[550]), .b(b[550]),
.c(c[550]));
    xtfa i551(.sum(sum[551]), .carry(carry[551]), .a(a[551]), .b(b[551]),
.c(c[551]));
    xtfa i552(.sum(sum[552]), .carry(carry[552]), .a(a[552]), .b(b[552]),
.c(c[552]));
    xtfa i553(.sum(sum[553]), .carry(carry[553]), .a(a[553]), .b(b[553]),
.c(c[553]));
    xtfa i554(.sum(sum[554]), .carry(carry[554]), .a(a[554]), .b(b[554]),
.c(c[554]));
    xtfa i555(.sum(sum[555]), .carry(carry[555]), .a(a[555]), .b(b[555]),
.c(c[555]));
    xtfa i556(.sum(sum[556]), .carry(carry[556]), .a(a[556]), .b(b[556]),
.c(c[556]));
    xtfa i557(.sum(sum[557]), .carry(carry[557]), .a(a[557]), .b(b[557]),
.c(c[557]));
    xtfa i558(.sum(sum[558]), .carry(carry[558]), .a(a[558]), .b(b[558]),
.c(c[558]));
    xtfa i559(.sum(sum[559]), .carry(carry[559]), .a(a[559]), .b(b[559]),
.c(c[559]));
    xtfa i560(.sum(sum[560]), .carry(carry[560]), .a(a[560]), .b(b[560]),
.c(c[560]));
```

```
    xtfa i561(.sum(sum[561]), .carry(carry[561]), .a(a[561]), .b(b[561]),
.c(c[561]));
    xtfa i562(.sum(sum[562]), .carry(carry[562]), .a(a[562]), .b(b[562]),
.c(c[562]));
    xtfa i563(.sum(sum[563]), .carry(carry[563]), .a(a[563]), .b(b[563]),
.c(c[563]));
    xtfa i564(.sum(sum[564]), .carry(carry[564]), .a(a[564]), .b(b[564]),
.c(c[564]));
    xtfa i565(.sum(sum[565]), .carry(carry[565]), .a(a[565]), .b(b[565]),
.c(c[565]));
    xtfa i566(.sum(sum[566]), .carry(carry[566]), .a(a[566]), .b(b[566]),
.c(c[566]));
    xtfa i567(.sum(sum[567]), .carry(carry[567]), .a(a[567]), .b(b[567]),
.c(c[567]));
    xtfa i568(.sum(sum[568]), .carry(carry[568]), .a(a[568]), .b(b[568]),
.c(c[568]));
    xtfa i569(.sum(sum[569]), .carry(carry[569]), .a(a[569]), .b(b[569]),
.c(c[569]));
    xtfa i570(.sum(sum[570]), .carry(carry[570]), .a(a[570]), .b(b[570]),
.c(c[570]));
    xtfa i571(.sum(sum[571]), .carry(carry[571]), .a(a[571]), .b(b[571]),
.c(c[571]));
    xtfa i572(.sum(sum[572]), .carry(carry[572]), .a(a[572]), .b(b[572]),
.c(c[572]));
    xtfa i573(.sum(sum[573]), .carry(carry[573]), .a(a[573]), .b(b[573]),
.c(c[573]));
    xtfa i574(.sum(sum[574]), .carry(carry[574]), .a(a[574]), .b(b[574]),
.c(c[574]));
    xtfa i575(.sum(sum[575]), .carry(carry[575]), .a(a[575]), .b(b[575]),
.c(c[575]));
    xtfa i576(.sum(sum[576]), .carry(carry[576]), .a(a[576]), .b(b[576]),
.c(c[576]));
    xtfa i577(.sum(sum[577]), .carry(carry[577]), .a(a[577]), .b(b[577]),
.c(c[577]));
    xtfa i578(.sum(sum[578]), .carry(carry[578]), .a(a[578]), .b(b[578]),
.c(c[578]));
    xtfa i579(.sum(sum[579]), .carry(carry[579]), .a(a[579]), .b(b[579]),
.c(c[579]));
    xtfa i580(.sum(sum[580]), .carry(carry[580]), .a(a[580]), .b(b[580]),
.c(c[580]));
    xtfa i581(.sum(sum[581]), .carry(carry[581]), .a(a[581]), .b(b[581]),
.c(c[581]));
    xtfa i582(.sum(sum[582]), .carry(carry[582]), .a(a[582]), .b(b[582]),
.c(c[582]));
    xtfa i583(.sum(sum[583]), .carry(carry[583]), .a(a[583]), .b(b[583]),
.c(c[583]));
    xtfa i584(.sum(sum[584]), .carry(carry[584]), .a(a[584]), .b(b[584]),
.c(c[584]));
    xtfa i585(.sum(sum[585]), .carry(carry[585]), .a(a[585]), .b(b[585]),
.c(c[585]));
    xtfa i586(.sum(sum[586]), .carry(carry[586]), .a(a[586]), .b(b[586]),
.c(c[586]));
    xtfa i587(.sum(sum[587]), .carry(carry[587]), .a(a[587]), .b(b[587]),
.c(c[587]));
    xtfa i588(.sum(sum[588]), .carry(carry[588]), .a(a[588]), .b(b[588]),
.c(c[588]));
```

```
    xtfa i589(.sum(sum[589]), .carry(carry[589]), .a(a[589]), .b(b[589]),
.c(c[589]));
    xtfa i590(.sum(sum[590]), .carry(carry[590]), .a(a[590]), .b(b[590]),
.c(c[590]));
    xtfa i591(.sum(sum[591]), .carry(carry[591]), .a(a[591]), .b(b[591]),
.c(c[591]));
    xtfa i592(.sum(sum[592]), .carry(carry[592]), .a(a[592]), .b(b[592]),
.c(c[592]));
    xtfa i593(.sum(sum[593]), .carry(carry[593]), .a(a[593]), .b(b[593]),
.c(c[593]));
    xtfa i594(.sum(sum[594]), .carry(carry[594]), .a(a[594]), .b(b[594]),
.c(c[594]));
    xtfa i595(.sum(sum[595]), .carry(carry[595]), .a(a[595]), .b(b[595]),
.c(c[595]));
    xtfa i596(.sum(sum[596]), .carry(carry[596]), .a(a[596]), .b(b[596]),
.c(c[596]));
    xtfa i597(.sum(sum[597]), .carry(carry[597]), .a(a[597]), .b(b[597]),
.c(c[597]));
    xtfa i598(.sum(sum[598]), .carry(carry[598]), .a(a[598]), .b(b[598]),
.c(c[598]));
    xtfa i599(.sum(sum[599]), .carry(carry[599]), .a(a[599]), .b(b[599]),
.c(c[599]));
    xtfa i600(.sum(sum[600]), .carry(carry[600]), .a(a[600]), .b(b[600]),
.c(c[600]));
    xtfa i601(.sum(sum[601]), .carry(carry[601]), .a(a[601]), .b(b[601]),
.c(c[601]));
    xtfa i602(.sum(sum[602]), .carry(carry[602]), .a(a[602]), .b(b[602]),
.c(c[602]));
    xtfa i603(.sum(sum[603]), .carry(carry[603]), .a(a[603]), .b(b[603]),
.c(c[603]));
    xtfa i604(.sum(sum[604]), .carry(carry[604]), .a(a[604]), .b(b[604]),
.c(c[604]));
    xtfa i605(.sum(sum[605]), .carry(carry[605]), .a(a[605]), .b(b[605]),
.c(c[605]));
    xtfa i606(.sum(sum[606]), .carry(carry[606]), .a(a[606]), .b(b[606]),
.c(c[606]));
    xtfa i607(.sum(sum[607]), .carry(carry[607]), .a(a[607]), .b(b[607]),
.c(c[607]));
    xtfa i608(.sum(sum[608]), .carry(carry[608]), .a(a[608]), .b(b[608]),
.c(c[608]));
    xtfa i609(.sum(sum[609]), .carry(carry[609]), .a(a[609]), .b(b[609]),
.c(c[609]));
    xtfa i610(.sum(sum[610]), .carry(carry[610]), .a(a[610]), .b(b[610]),
.c(c[610]));
    xtfa i611(.sum(sum[611]), .carry(carry[611]), .a(a[611]), .b(b[611]),
.c(c[611]));
    xtfa i612(.sum(sum[612]), .carry(carry[612]), .a(a[612]), .b(b[612]),
.c(c[612]));
    xtfa i613(.sum(sum[613]), .carry(carry[613]), .a(a[613]), .b(b[613]),
.c(c[613]));
    xtfa i614(.sum(sum[614]), .carry(carry[614]), .a(a[614]), .b(b[614]),
.c(c[614]));
    xtfa i615(.sum(sum[615]), .carry(carry[615]), .a(a[615]), .b(b[615]),
.c(c[615]));
    xtfa i616(.sum(sum[616]), .carry(carry[616]), .a(a[616]), .b(b[616]),
.c(c[616]));
```

```
    xtfa i617(.sum(sum[617]), .carry(carry[617]), .a(a[617]), .b(b[617]),
.c(c[617]));
    xtfa i618(.sum(sum[618]), .carry(carry[618]), .a(a[618]), .b(b[618]),
.c(c[618]));
    xtfa i619(.sum(sum[619]), .carry(carry[619]), .a(a[619]), .b(b[619]),
.c(c[619]));
    xtfa i620(.sum(sum[620]), .carry(carry[620]), .a(a[620]), .b(b[620]),
.c(c[620]));
    xtfa i621(.sum(sum[621]), .carry(carry[621]), .a(a[621]), .b(b[621]),
.c(c[621]));
    xtfa i622(.sum(sum[622]), .carry(carry[622]), .a(a[622]), .b(b[622]),
.c(c[622]));
    xtfa i623(.sum(sum[623]), .carry(carry[623]), .a(a[623]), .b(b[623]),
.c(c[623]));
    xtfa i624(.sum(sum[624]), .carry(carry[624]), .a(a[624]), .b(b[624]),
.c(c[624]));
    xtfa i625(.sum(sum[625]), .carry(carry[625]), .a(a[625]), .b(b[625]),
.c(c[625]));
    xtfa i626(.sum(sum[626]), .carry(carry[626]), .a(a[626]), .b(b[626]),
.c(c[626]));
    xtfa i627(.sum(sum[627]), .carry(carry[627]), .a(a[627]), .b(b[627]),
.c(c[627]));
    xtfa i628(.sum(sum[628]), .carry(carry[628]), .a(a[628]), .b(b[628]),
.c(c[628]));
    xtfa i629(.sum(sum[629]), .carry(carry[629]), .a(a[629]), .b(b[629]),
.c(c[629]));
    xtfa i630(.sum(sum[630]), .carry(carry[630]), .a(a[630]), .b(b[630]),
.c(c[630]));
    xtfa i631(.sum(sum[631]), .carry(carry[631]), .a(a[631]), .b(b[631]),
.c(c[631]));
    xtfa i632(.sum(sum[632]), .carry(carry[632]), .a(a[632]), .b(b[632]),
.c(c[632]));
    xtfa i633(.sum(sum[633]), .carry(carry[633]), .a(a[633]), .b(b[633]),
.c(c[633]));
    xtfa i634(.sum(sum[634]), .carry(carry[634]), .a(a[634]), .b(b[634]),
.c(c[634]));
    xtfa i635(.sum(sum[635]), .carry(carry[635]), .a(a[635]), .b(b[635]),
.c(c[635]));
    xtfa i636(.sum(sum[636]), .carry(carry[636]), .a(a[636]), .b(b[636]),
.c(c[636]));
    xtfa i637(.sum(sum[637]), .carry(carry[637]), .a(a[637]), .b(b[637]),
.c(c[637]));
    xtfa i638(.sum(sum[638]), .carry(carry[638]), .a(a[638]), .b(b[638]),
.c(c[638]));
    xtfa i639(.sum(sum[639]), .carry(carry[639]), .a(a[639]), .b(b[639]),
.c(c[639]));
    xtfa i640(.sum(sum[640]), .carry(carry[640]), .a(a[640]), .b(b[640]),
.c(c[640]));
    xtfa i641(.sum(sum[641]), .carry(carry[641]), .a(a[641]), .b(b[641]),
.c(c[641]));
    xtfa i642(.sum(sum[642]), .carry(carry[642]), .a(a[642]), .b(b[642]),
.c(c[642]));
    xtfa i643(.sum(sum[643]), .carry(carry[643]), .a(a[643]), .b(b[643]),
.c(c[643]));
    xtfa i644(.sum(sum[644]), .carry(carry[644]), .a(a[644]), .b(b[644]),
.c(c[644]));
```

```
    xtfa i645(.sum(sum[645]), .carry(carry[645]), .a(a[645]), .b(b[645]),
.c(c[645]));
    xtfa i646(.sum(sum[646]), .carry(carry[646]), .a(a[646]), .b(b[646]),
.c(c[646]));
    xtfa i647(.sum(sum[647]), .carry(carry[647]), .a(a[647]), .b(b[647]),
.c(c[647]));
    xtfa i648(.sum(sum[648]), .carry(carry[648]), .a(a[648]), .b(b[648]),
.c(c[648]));
    xtfa i649(.sum(sum[649]), .carry(carry[649]), .a(a[649]), .b(b[649]),
.c(c[649]));
    xtfa i650(.sum(sum[650]), .carry(carry[650]), .a(a[650]), .b(b[650]),
.c(c[650]));
    xtfa i651(.sum(sum[651]), .carry(carry[651]), .a(a[651]), .b(b[651]),
.c(c[651]));
    xtfa i652(.sum(sum[652]), .carry(carry[652]), .a(a[652]), .b(b[652]),
.c(c[652]));
    xtfa i653(.sum(sum[653]), .carry(carry[653]), .a(a[653]), .b(b[653]),
.c(c[653]));
    xtfa i654(.sum(sum[654]), .carry(carry[654]), .a(a[654]), .b(b[654]),
.c(c[654]));
    xtfa i655(.sum(sum[655]), .carry(carry[655]), .a(a[655]), .b(b[655]),
.c(c[655]));
    xtfa i656(.sum(sum[656]), .carry(carry[656]), .a(a[656]), .b(b[656]),
.c(c[656]));
    xtfa i657(.sum(sum[657]), .carry(carry[657]), .a(a[657]), .b(b[657]),
.c(c[657]));
    xtfa i658(.sum(sum[658]), .carry(carry[658]), .a(a[658]), .b(b[658]),
.c(c[658]));
    xtfa i659(.sum(sum[659]), .carry(carry[659]), .a(a[659]), .b(b[659]),
.c(c[659]));
    xtfa i660(.sum(sum[660]), .carry(carry[660]), .a(a[660]), .b(b[660]),
.c(c[660]));
    xtfa i661(.sum(sum[661]), .carry(carry[661]), .a(a[661]), .b(b[661]),
.c(c[661]));
    xtfa i662(.sum(sum[662]), .carry(carry[662]), .a(a[662]), .b(b[662]),
.c(c[662]));
    xtfa i663(.sum(sum[663]), .carry(carry[663]), .a(a[663]), .b(b[663]),
.c(c[663]));
    xtfa i664(.sum(sum[664]), .carry(carry[664]), .a(a[664]), .b(b[664]),
.c(c[664]));
    xtfa i665(.sum(sum[665]), .carry(carry[665]), .a(a[665]), .b(b[665]),
.c(c[665]));
    xtfa i666(.sum(sum[666]), .carry(carry[666]), .a(a[666]), .b(b[666]),
.c(c[666]));
    xtfa i667(.sum(sum[667]), .carry(carry[667]), .a(a[667]), .b(b[667]),
.c(c[667]));
    xtfa i668(.sum(sum[668]), .carry(carry[668]), .a(a[668]), .b(b[668]),
.c(c[668]));
    xtfa i669(.sum(sum[669]), .carry(carry[669]), .a(a[669]), .b(b[669]),
.c(c[669]));
    xtfa i670(.sum(sum[670]), .carry(carry[670]), .a(a[670]), .b(b[670]),
.c(c[670]));
    xtfa i671(.sum(sum[671]), .carry(carry[671]), .a(a[671]), .b(b[671]),
.c(c[671]));
    xtfa i672(.sum(sum[672]), .carry(carry[672]), .a(a[672]), .b(b[672]),
.c(c[672]));
```

```
    xtfa i673(.sum(sum[673]), .carry(carry[673]), .a(a[673]), .b(b[673]),
.c(c[673]));
    xtfa i674(.sum(sum[674]), .carry(carry[674]), .a(a[674]), .b(b[674]),
.c(c[674]));
    xtfa i675(.sum(sum[675]), .carry(carry[675]), .a(a[675]), .b(b[675]),
.c(c[675]));
    xtfa i676(.sum(sum[676]), .carry(carry[676]), .a(a[676]), .b(b[676]),
.c(c[676]));
    xtfa i677(.sum(sum[677]), .carry(carry[677]), .a(a[677]), .b(b[677]),
.c(c[677]));
    xtfa i678(.sum(sum[678]), .carry(carry[678]), .a(a[678]), .b(b[678]),
.c(c[678]));
    xtfa i679(.sum(sum[679]), .carry(carry[679]), .a(a[679]), .b(b[679]),
.c(c[679]));
    xtfa i680(.sum(sum[680]), .carry(carry[680]), .a(a[680]), .b(b[680]),
.c(c[680]));
    xtfa i681(.sum(sum[681]), .carry(carry[681]), .a(a[681]), .b(b[681]),
.c(c[681]));
    xtfa i682(.sum(sum[682]), .carry(carry[682]), .a(a[682]), .b(b[682]),
.c(c[682]));
    xtfa i683(.sum(sum[683]), .carry(carry[683]), .a(a[683]), .b(b[683]),
.c(c[683]));
    xtfa i684(.sum(sum[684]), .carry(carry[684]), .a(a[684]), .b(b[684]),
.c(c[684]));
    xtfa i685(.sum(sum[685]), .carry(carry[685]), .a(a[685]), .b(b[685]),
.c(c[685]));
    xtfa i686(.sum(sum[686]), .carry(carry[686]), .a(a[686]), .b(b[686]),
.c(c[686]));
    xtfa i687(.sum(sum[687]), .carry(carry[687]), .a(a[687]), .b(b[687]),
.c(c[687]));
    xtfa i688(.sum(sum[688]), .carry(carry[688]), .a(a[688]), .b(b[688]),
.c(c[688]));
    xtfa i689(.sum(sum[689]), .carry(carry[689]), .a(a[689]), .b(b[689]),
.c(c[689]));
    xtfa i690(.sum(sum[690]), .carry(carry[690]), .a(a[690]), .b(b[690]),
.c(c[690]));
    xtfa i691(.sum(sum[691]), .carry(carry[691]), .a(a[691]), .b(b[691]),
.c(c[691]));
    xtfa i692(.sum(sum[692]), .carry(carry[692]), .a(a[692]), .b(b[692]),
.c(c[692]));
    xtfa i693(.sum(sum[693]), .carry(carry[693]), .a(a[693]), .b(b[693]),
.c(c[693]));
    xtfa i694(.sum(sum[694]), .carry(carry[694]), .a(a[694]), .b(b[694]),
.c(c[694]));
    xtfa i695(.sum(sum[695]), .carry(carry[695]), .a(a[695]), .b(b[695]),
.c(c[695]));
    xtfa i696(.sum(sum[696]), .carry(carry[696]), .a(a[696]), .b(b[696]),
.c(c[696]));
    xtfa i697(.sum(sum[697]), .carry(carry[697]), .a(a[697]), .b(b[697]),
.c(c[697]));
    xtfa i698(.sum(sum[698]), .carry(carry[698]), .a(a[698]), .b(b[698]),
.c(c[698]));
    xtfa i699(.sum(sum[699]), .carry(carry[699]), .a(a[699]), .b(b[699]),
.c(c[699]));
    xtfa i700(.sum(sum[700]), .carry(carry[700]), .a(a[700]), .b(b[700]),
.c(c[700]));
```

```
    xtfa i701(.sum(sum[701]), .carry(carry[701]), .a(a[701]), .b(b[701]),
.c(c[701]));
    xtfa i702(.sum(sum[702]), .carry(carry[702]), .a(a[702]), .b(b[702]),
.c(c[702]));
    xtfa i703(.sum(sum[703]), .carry(carry[703]), .a(a[703]), .b(b[703]),
.c(c[703]));
    xtfa i704(.sum(sum[704]), .carry(carry[704]), .a(a[704]), .b(b[704]),
.c(c[704]));
    xtfa i705(.sum(sum[705]), .carry(carry[705]), .a(a[705]), .b(b[705]),
.c(c[705]));
    xtfa i706(.sum(sum[706]), .carry(carry[706]), .a(a[706]), .b(b[706]),
.c(c[706]));
    xtfa i707(.sum(sum[707]), .carry(carry[707]), .a(a[707]), .b(b[707]),
.c(c[707]));
    xtfa i708(.sum(sum[708]), .carry(carry[708]), .a(a[708]), .b(b[708]),
.c(c[708]));
    xtfa i709(.sum(sum[709]), .carry(carry[709]), .a(a[709]), .b(b[709]),
.c(c[709]));
    xtfa i710(.sum(sum[710]), .carry(carry[710]), .a(a[710]), .b(b[710]),
.c(c[710]));
    xtfa i711(.sum(sum[711]), .carry(carry[711]), .a(a[711]), .b(b[711]),
.c(c[711]));
    xtfa i712(.sum(sum[712]), .carry(carry[712]), .a(a[712]), .b(b[712]),
.c(c[712]));
    xtfa i713(.sum(sum[713]), .carry(carry[713]), .a(a[713]), .b(b[713]),
.c(c[713]));
    xtfa i714(.sum(sum[714]), .carry(carry[714]), .a(a[714]), .b(b[714]),
.c(c[714]));
    xtfa i715(.sum(sum[715]), .carry(carry[715]), .a(a[715]), .b(b[715]),
.c(c[715]));
    xtfa i716(.sum(sum[716]), .carry(carry[716]), .a(a[716]), .b(b[716]),
.c(c[716]));
    xtfa i717(.sum(sum[717]), .carry(carry[717]), .a(a[717]), .b(b[717]),
.c(c[717]));
    xtfa i718(.sum(sum[718]), .carry(carry[718]), .a(a[718]), .b(b[718]),
.c(c[718]));
    xtfa i719(.sum(sum[719]), .carry(carry[719]), .a(a[719]), .b(b[719]),
.c(c[719]));
    xtfa i720(.sum(sum[720]), .carry(carry[720]), .a(a[720]), .b(b[720]),
.c(c[720]));
    xtfa i721(.sum(sum[721]), .carry(carry[721]), .a(a[721]), .b(b[721]),
.c(c[721]));
    xtfa i722(.sum(sum[722]), .carry(carry[722]), .a(a[722]), .b(b[722]),
.c(c[722]));
    xtfa i723(.sum(sum[723]), .carry(carry[723]), .a(a[723]), .b(b[723]),
.c(c[723]));
    xtfa i724(.sum(sum[724]), .carry(carry[724]), .a(a[724]), .b(b[724]),
.c(c[724]));
    xtfa i725(.sum(sum[725]), .carry(carry[725]), .a(a[725]), .b(b[725]),
.c(c[725]));
    xtfa i726(.sum(sum[726]), .carry(carry[726]), .a(a[726]), .b(b[726]),
.c(c[726]));
    xtfa i727(.sum(sum[727]), .carry(carry[727]), .a(a[727]), .b(b[727]),
.c(c[727]));
    xtfa i728(.sum(sum[728]), .carry(carry[728]), .a(a[728]), .b(b[728]),
.c(c[728]));
```

```
    xtfa i729(.sum(sum[729]), .carry(carry[729]), .a(a[729]), .b(b[729]),
.c(c[729]));
    xtfa i730(.sum(sum[730]), .carry(carry[730]), .a(a[730]), .b(b[730]),
.c(c[730]));
    xtfa i731(.sum(sum[731]), .carry(carry[731]), .a(a[731]), .b(b[731]),
.c(c[731]));
    xtfa i732(.sum(sum[732]), .carry(carry[732]), .a(a[732]), .b(b[732]),
.c(c[732]));
    xtfa i733(.sum(sum[733]), .carry(carry[733]), .a(a[733]), .b(b[733]),
.c(c[733]));
    xtfa i734(.sum(sum[734]), .carry(carry[734]), .a(a[734]), .b(b[734]),
.c(c[734]));
    xtfa i735(.sum(sum[735]), .carry(carry[735]), .a(a[735]), .b(b[735]),
.c(c[735]));
    xtfa i736(.sum(sum[736]), .carry(carry[736]), .a(a[736]), .b(b[736]),
.c(c[736]));
    xtfa i737(.sum(sum[737]), .carry(carry[737]), .a(a[737]), .b(b[737]),
.c(c[737]));
    xtfa i738(.sum(sum[738]), .carry(carry[738]), .a(a[738]), .b(b[738]),
.c(c[738]));
    xtfa i739(.sum(sum[739]), .carry(carry[739]), .a(a[739]), .b(b[739]),
.c(c[739]));
    xtfa i740(.sum(sum[740]), .carry(carry[740]), .a(a[740]), .b(b[740]),
.c(c[740]));
    xtfa i741(.sum(sum[741]), .carry(carry[741]), .a(a[741]), .b(b[741]),
.c(c[741]));
    xtfa i742(.sum(sum[742]), .carry(carry[742]), .a(a[742]), .b(b[742]),
.c(c[742]));
    xtfa i743(.sum(sum[743]), .carry(carry[743]), .a(a[743]), .b(b[743]),
.c(c[743]));
    xtfa i744(.sum(sum[744]), .carry(carry[744]), .a(a[744]), .b(b[744]),
.c(c[744]));
    xtfa i745(.sum(sum[745]), .carry(carry[745]), .a(a[745]), .b(b[745]),
.c(c[745]));
    xtfa i746(.sum(sum[746]), .carry(carry[746]), .a(a[746]), .b(b[746]),
.c(c[746]));
    xtfa i747(.sum(sum[747]), .carry(carry[747]), .a(a[747]), .b(b[747]),
.c(c[747]));
    xtfa i748(.sum(sum[748]), .carry(carry[748]), .a(a[748]), .b(b[748]),
.c(c[748]));
    xtfa i749(.sum(sum[749]), .carry(carry[749]), .a(a[749]), .b(b[749]),
.c(c[749]));
    xtfa i750(.sum(sum[750]), .carry(carry[750]), .a(a[750]), .b(b[750]),
.c(c[750]));
    xtfa i751(.sum(sum[751]), .carry(carry[751]), .a(a[751]), .b(b[751]),
.c(c[751]));
    xtfa i752(.sum(sum[752]), .carry(carry[752]), .a(a[752]), .b(b[752]),
.c(c[752]));
    xtfa i753(.sum(sum[753]), .carry(carry[753]), .a(a[753]), .b(b[753]),
.c(c[753]));
    xtfa i754(.sum(sum[754]), .carry(carry[754]), .a(a[754]), .b(b[754]),
.c(c[754]));
    xtfa i755(.sum(sum[755]), .carry(carry[755]), .a(a[755]), .b(b[755]),
.c(c[755]));
    xtfa i756(.sum(sum[756]), .carry(carry[756]), .a(a[756]), .b(b[756]),
.c(c[756]));
```

```
    xtfa i757(.sum(sum[757]), .carry(carry[757]), .a(a[757]), .b(b[757]),
.c(c[757]));
    xtfa i758(.sum(sum[758]), .carry(carry[758]), .a(a[758]), .b(b[758]),
.c(c[758]));
    xtfa i759(.sum(sum[759]), .carry(carry[759]), .a(a[759]), .b(b[759]),
.c(c[759]));
    xtfa i760(.sum(sum[760]), .carry(carry[760]), .a(a[760]), .b(b[760]),
.c(c[760]));
    xtfa i761(.sum(sum[761]), .carry(carry[761]), .a(a[761]), .b(b[761]),
.c(c[761]));
    xtfa i762(.sum(sum[762]), .carry(carry[762]), .a(a[762]), .b(b[762]),
.c(c[762]));
    xtfa i763(.sum(sum[763]), .carry(carry[763]), .a(a[763]), .b(b[763]),
.c(c[763]));
    xtfa i764(.sum(sum[764]), .carry(carry[764]), .a(a[764]), .b(b[764]),
.c(c[764]));
    xtfa i765(.sum(sum[765]), .carry(carry[765]), .a(a[765]), .b(b[765]),
.c(c[765]));
    xtfa i766(.sum(sum[766]), .carry(carry[766]), .a(a[766]), .b(b[766]),
.c(c[766]));
    xtfa i767(.sum(sum[767]), .carry(carry[767]), .a(a[767]), .b(b[767]),
.c(c[767]));
    xtfa i768(.sum(sum[768]), .carry(carry[768]), .a(a[768]), .b(b[768]),
.c(c[768]));
    xtfa i769(.sum(sum[769]), .carry(carry[769]), .a(a[769]), .b(b[769]),
.c(c[769]));
    xtfa i770(.sum(sum[770]), .carry(carry[770]), .a(a[770]), .b(b[770]),
.c(c[770]));
    xtfa i771(.sum(sum[771]), .carry(carry[771]), .a(a[771]), .b(b[771]),
.c(c[771]));
    xtfa i772(.sum(sum[772]), .carry(carry[772]), .a(a[772]), .b(b[772]),
.c(c[772]));
    xtfa i773(.sum(sum[773]), .carry(carry[773]), .a(a[773]), .b(b[773]),
.c(c[773]));
    xtfa i774(.sum(sum[774]), .carry(carry[774]), .a(a[774]), .b(b[774]),
.c(c[774]));
    xtfa i775(.sum(sum[775]), .carry(carry[775]), .a(a[775]), .b(b[775]),
.c(c[775]));
    xtfa i776(.sum(sum[776]), .carry(carry[776]), .a(a[776]), .b(b[776]),
.c(c[776]));
    xtfa i777(.sum(sum[777]), .carry(carry[777]), .a(a[777]), .b(b[777]),
.c(c[777]));
    xtfa i778(.sum(sum[778]), .carry(carry[778]), .a(a[778]), .b(b[778]),
.c(c[778]));
    xtfa i779(.sum(sum[779]), .carry(carry[779]), .a(a[779]), .b(b[779]),
.c(c[779]));
    xtfa i780(.sum(sum[780]), .carry(carry[780]), .a(a[780]), .b(b[780]),
.c(c[780]));
    xtfa i781(.sum(sum[781]), .carry(carry[781]), .a(a[781]), .b(b[781]),
.c(c[781]));
    xtfa i782(.sum(sum[782]), .carry(carry[782]), .a(a[782]), .b(b[782]),
.c(c[782]));
    xtfa i783(.sum(sum[783]), .carry(carry[783]), .a(a[783]), .b(b[783]),
.c(c[783]));
    xtfa i784(.sum(sum[784]), .carry(carry[784]), .a(a[784]), .b(b[784]),
.c(c[784]));
```

```
    xtfa i785(.sum(sum[785]), .carry(carry[785]), .a(a[785]), .b(b[785]),
.c(c[785]));
    xtfa i786(.sum(sum[786]), .carry(carry[786]), .a(a[786]), .b(b[786]),
.c(c[786]));
    xtfa i787(.sum(sum[787]), .carry(carry[787]), .a(a[787]), .b(b[787]),
.c(c[787]));
    xtfa i788(.sum(sum[788]), .carry(carry[788]), .a(a[788]), .b(b[788]),
.c(c[788]));
    xtfa i789(.sum(sum[789]), .carry(carry[789]), .a(a[789]), .b(b[789]),
.c(c[789]));
    xtfa i790(.sum(sum[790]), .carry(carry[790]), .a(a[790]), .b(b[790]),
.c(c[790]));
    xtfa i791(.sum(sum[791]), .carry(carry[791]), .a(a[791]), .b(b[791]),
.c(c[791]));
    xtfa i792(.sum(sum[792]), .carry(carry[792]), .a(a[792]), .b(b[792]),
.c(c[792]));
    xtfa i793(.sum(sum[793]), .carry(carry[793]), .a(a[793]), .b(b[793]),
.c(c[793]));
    xtfa i794(.sum(sum[794]), .carry(carry[794]), .a(a[794]), .b(b[794]),
.c(c[794]));
    xtfa i795(.sum(sum[795]), .carry(carry[795]), .a(a[795]), .b(b[795]),
.c(c[795]));
    xtfa i796(.sum(sum[796]), .carry(carry[796]), .a(a[796]), .b(b[796]),
.c(c[796]));
    xtfa i797(.sum(sum[797]), .carry(carry[797]), .a(a[797]), .b(b[797]),
.c(c[797]));
    xtfa i798(.sum(sum[798]), .carry(carry[798]), .a(a[798]), .b(b[798]),
.c(c[798]));
    xtfa i799(.sum(sum[799]), .carry(carry[799]), .a(a[799]), .b(b[799]),
.c(c[799]));
    xtfa i800(.sum(sum[800]), .carry(carry[800]), .a(a[800]), .b(b[800]),
.c(c[800]));
    xtfa i801(.sum(sum[801]), .carry(carry[801]), .a(a[801]), .b(b[801]),
.c(c[801]));
    xtfa i802(.sum(sum[802]), .carry(carry[802]), .a(a[802]), .b(b[802]),
.c(c[802]));
    xtfa i803(.sum(sum[803]), .carry(carry[803]), .a(a[803]), .b(b[803]),
.c(c[803]));
    xtfa i804(.sum(sum[804]), .carry(carry[804]), .a(a[804]), .b(b[804]),
.c(c[804]));
    xtfa i805(.sum(sum[805]), .carry(carry[805]), .a(a[805]), .b(b[805]),
.c(c[805]));
    xtfa i806(.sum(sum[806]), .carry(carry[806]), .a(a[806]), .b(b[806]),
.c(c[806]));
    xtfa i807(.sum(sum[807]), .carry(carry[807]), .a(a[807]), .b(b[807]),
.c(c[807]));
    xtfa i808(.sum(sum[808]), .carry(carry[808]), .a(a[808]), .b(b[808]),
.c(c[808]));
    xtfa i809(.sum(sum[809]), .carry(carry[809]), .a(a[809]), .b(b[809]),
.c(c[809]));
    xtfa i810(.sum(sum[810]), .carry(carry[810]), .a(a[810]), .b(b[810]),
.c(c[810]));
    xtfa i811(.sum(sum[811]), .carry(carry[811]), .a(a[811]), .b(b[811]),
.c(c[811]));
    xtfa i812(.sum(sum[812]), .carry(carry[812]), .a(a[812]), .b(b[812]),
.c(c[812]));
```

```
    xtfa i813(.sum(sum[813]), .carry(carry[813]), .a(a[813]), .b(b[813]),
.c(c[813]));
    xtfa i814(.sum(sum[814]), .carry(carry[814]), .a(a[814]), .b(b[814]),
.c(c[814]));
    xtfa i815(.sum(sum[815]), .carry(carry[815]), .a(a[815]), .b(b[815]),
.c(c[815]));
    xtfa i816(.sum(sum[816]), .carry(carry[816]), .a(a[816]), .b(b[816]),
.c(c[816]));
    xtfa i817(.sum(sum[817]), .carry(carry[817]), .a(a[817]), .b(b[817]),
.c(c[817]));
    xtfa i818(.sum(sum[818]), .carry(carry[818]), .a(a[818]), .b(b[818]),
.c(c[818]));
    xtfa i819(.sum(sum[819]), .carry(carry[819]), .a(a[819]), .b(b[819]),
.c(c[819]));
    xtfa i820(.sum(sum[820]), .carry(carry[820]), .a(a[820]), .b(b[820]),
.c(c[820]));
    xtfa i821(.sum(sum[821]), .carry(carry[821]), .a(a[821]), .b(b[821]),
.c(c[821]));
    xtfa i822(.sum(sum[822]), .carry(carry[822]), .a(a[822]), .b(b[822]),
.c(c[822]));
    xtfa i823(.sum(sum[823]), .carry(carry[823]), .a(a[823]), .b(b[823]),
.c(c[823]));
    xtfa i824(.sum(sum[824]), .carry(carry[824]), .a(a[824]), .b(b[824]),
.c(c[824]));
    xtfa i825(.sum(sum[825]), .carry(carry[825]), .a(a[825]), .b(b[825]),
.c(c[825]));
    xtfa i826(.sum(sum[826]), .carry(carry[826]), .a(a[826]), .b(b[826]),
.c(c[826]));
    xtfa i827(.sum(sum[827]), .carry(carry[827]), .a(a[827]), .b(b[827]),
.c(c[827]));
    xtfa i828(.sum(sum[828]), .carry(carry[828]), .a(a[828]), .b(b[828]),
.c(c[828]));
    xtfa i829(.sum(sum[829]), .carry(carry[829]), .a(a[829]), .b(b[829]),
.c(c[829]));
    xtfa i830(.sum(sum[830]), .carry(carry[830]), .a(a[830]), .b(b[830]),
.c(c[830]));
    xtfa i831(.sum(sum[831]), .carry(carry[831]), .a(a[831]), .b(b[831]),
.c(c[831]));
    xtfa i832(.sum(sum[832]), .carry(carry[832]), .a(a[832]), .b(b[832]),
.c(c[832]));
    xtfa i833(.sum(sum[833]), .carry(carry[833]), .a(a[833]), .b(b[833]),
.c(c[833]));
    xtfa i834(.sum(sum[834]), .carry(carry[834]), .a(a[834]), .b(b[834]),
.c(c[834]));
    xtfa i835(.sum(sum[835]), .carry(carry[835]), .a(a[835]), .b(b[835]),
.c(c[835]));
    xtfa i836(.sum(sum[836]), .carry(carry[836]), .a(a[836]), .b(b[836]),
.c(c[836]));
    xtfa i837(.sum(sum[837]), .carry(carry[837]), .a(a[837]), .b(b[837]),
.c(c[837]));
    xtfa i838(.sum(sum[838]), .carry(carry[838]), .a(a[838]), .b(b[838]),
.c(c[838]));
    xtfa i839(.sum(sum[839]), .carry(carry[839]), .a(a[839]), .b(b[839]),
.c(c[839]));
    xtfa i840(.sum(sum[840]), .carry(carry[840]), .a(a[840]), .b(b[840]),
.c(c[840]));
```

```
    xtfa i841(.sum(sum[841]), .carry(carry[841]), .a(a[841]), .b(b[841]),
.c(c[841]));
    xtfa i842(.sum(sum[842]), .carry(carry[842]), .a(a[842]), .b(b[842]),
.c(c[842]));
    xtfa i843(.sum(sum[843]), .carry(carry[843]), .a(a[843]), .b(b[843]),
.c(c[843]));
    xtfa i844(.sum(sum[844]), .carry(carry[844]), .a(a[844]), .b(b[844]),
.c(c[844]));
    xtfa i845(.sum(sum[845]), .carry(carry[845]), .a(a[845]), .b(b[845]),
.c(c[845]));
    xtfa i846(.sum(sum[846]), .carry(carry[846]), .a(a[846]), .b(b[846]),
.c(c[846]));
    xtfa i847(.sum(sum[847]), .carry(carry[847]), .a(a[847]), .b(b[847]),
.c(c[847]));
    xtfa i848(.sum(sum[848]), .carry(carry[848]), .a(a[848]), .b(b[848]),
.c(c[848]));
    xtfa i849(.sum(sum[849]), .carry(carry[849]), .a(a[849]), .b(b[849]),
.c(c[849]));
    xtfa i850(.sum(sum[850]), .carry(carry[850]), .a(a[850]), .b(b[850]),
.c(c[850]));
    xtfa i851(.sum(sum[851]), .carry(carry[851]), .a(a[851]), .b(b[851]),
.c(c[851]));
    xtfa i852(.sum(sum[852]), .carry(carry[852]), .a(a[852]), .b(b[852]),
.c(c[852]));
    xtfa i853(.sum(sum[853]), .carry(carry[853]), .a(a[853]), .b(b[853]),
.c(c[853]));
    xtfa i854(.sum(sum[854]), .carry(carry[854]), .a(a[854]), .b(b[854]),
.c(c[854]));
    xtfa i855(.sum(sum[855]), .carry(carry[855]), .a(a[855]), .b(b[855]),
.c(c[855]));
    xtfa i856(.sum(sum[856]), .carry(carry[856]), .a(a[856]), .b(b[856]),
.c(c[856]));
    xtfa i857(.sum(sum[857]), .carry(carry[857]), .a(a[857]), .b(b[857]),
.c(c[857]));
    xtfa i858(.sum(sum[858]), .carry(carry[858]), .a(a[858]), .b(b[858]),
.c(c[858]));
    xtfa i859(.sum(sum[859]), .carry(carry[859]), .a(a[859]), .b(b[859]),
.c(c[859]));
    xtfa i860(.sum(sum[860]), .carry(carry[860]), .a(a[860]), .b(b[860]),
.c(c[860]));
    xtfa i861(.sum(sum[861]), .carry(carry[861]), .a(a[861]), .b(b[861]),
.c(c[861]));
    xtfa i862(.sum(sum[862]), .carry(carry[862]), .a(a[862]), .b(b[862]),
.c(c[862]));
    xtfa i863(.sum(sum[863]), .carry(carry[863]), .a(a[863]), .b(b[863]),
.c(c[863]));
    xtfa i864(.sum(sum[864]), .carry(carry[864]), .a(a[864]), .b(b[864]),
.c(c[864]));
    xtfa i865(.sum(sum[865]), .carry(carry[865]), .a(a[865]), .b(b[865]),
.c(c[865]));
    xtfa i866(.sum(sum[866]), .carry(carry[866]), .a(a[866]), .b(b[866]),
.c(c[866]));
    xtfa i867(.sum(sum[867]), .carry(carry[867]), .a(a[867]), .b(b[867]),
.c(c[867]));
    xtfa i868(.sum(sum[868]), .carry(carry[868]), .a(a[868]), .b(b[868]),
.c(c[868]));
```

```
    xtfa i869(.sum(sum[869]), .carry(carry[869]), .a(a[869]), .b(b[869]),
.c(c[869]));
    xtfa i870(.sum(sum[870]), .carry(carry[870]), .a(a[870]), .b(b[870]),
.c(c[870]));
    xtfa i871(.sum(sum[871]), .carry(carry[871]), .a(a[871]), .b(b[871]),
.c(c[871]));
    xtfa i872(.sum(sum[872]), .carry(carry[872]), .a(a[872]), .b(b[872]),
.c(c[872]));
    xtfa i873(.sum(sum[873]), .carry(carry[873]), .a(a[873]), .b(b[873]),
.c(c[873]));
    xtfa i874(.sum(sum[874]), .carry(carry[874]), .a(a[874]), .b(b[874]),
.c(c[874]));
    xtfa i875(.sum(sum[875]), .carry(carry[875]), .a(a[875]), .b(b[875]),
.c(c[875]));
    xtfa i876(.sum(sum[876]), .carry(carry[876]), .a(a[876]), .b(b[876]),
.c(c[876]));
    xtfa i877(.sum(sum[877]), .carry(carry[877]), .a(a[877]), .b(b[877]),
.c(c[877]));
    xtfa i878(.sum(sum[878]), .carry(carry[878]), .a(a[878]), .b(b[878]),
.c(c[878]));
    xtfa i879(.sum(sum[879]), .carry(carry[879]), .a(a[879]), .b(b[879]),
.c(c[879]));
    xtfa i880(.sum(sum[880]), .carry(carry[880]), .a(a[880]), .b(b[880]),
.c(c[880]));
    xtfa i881(.sum(sum[881]), .carry(carry[881]), .a(a[881]), .b(b[881]),
.c(c[881]));
    xtfa i882(.sum(sum[882]), .carry(carry[882]), .a(a[882]), .b(b[882]),
.c(c[882]));
    xtfa i883(.sum(sum[883]), .carry(carry[883]), .a(a[883]), .b(b[883]),
.c(c[883]));
    xtfa i884(.sum(sum[884]), .carry(carry[884]), .a(a[884]), .b(b[884]),
.c(c[884]));
    xtfa i885(.sum(sum[885]), .carry(carry[885]), .a(a[885]), .b(b[885]),
.c(c[885]));
    xtfa i886(.sum(sum[886]), .carry(carry[886]), .a(a[886]), .b(b[886]),
.c(c[886]));
    xtfa i887(.sum(sum[887]), .carry(carry[887]), .a(a[887]), .b(b[887]),
.c(c[887]));
    xtfa i888(.sum(sum[888]), .carry(carry[888]), .a(a[888]), .b(b[888]),
.c(c[888]));
    xtfa i889(.sum(sum[889]), .carry(carry[889]), .a(a[889]), .b(b[889]),
.c(c[889]));
    xtfa i890(.sum(sum[890]), .carry(carry[890]), .a(a[890]), .b(b[890]),
.c(c[890]));
    xtfa i891(.sum(sum[891]), .carry(carry[891]), .a(a[891]), .b(b[891]),
.c(c[891]));
    xtfa i892(.sum(sum[892]), .carry(carry[892]), .a(a[892]), .b(b[892]),
.c(c[892]));
    xtfa i893(.sum(sum[893]), .carry(carry[893]), .a(a[893]), .b(b[893]),
.c(c[893]));
    xtfa i894(.sum(sum[894]), .carry(carry[894]), .a(a[894]), .b(b[894]),
.c(c[894]));
    xtfa i895(.sum(sum[895]), .carry(carry[895]), .a(a[895]), .b(b[895]),
.c(c[895]));
    xtfa i896(.sum(sum[896]), .carry(carry[896]), .a(a[896]), .b(b[896]),
.c(c[896]));
```

```
    xtfa i897(.sum(sum[897]), .carry(carry[897]), .a(a[897]), .b(b[897]),
.c(c[897]));
    xtfa i898(.sum(sum[898]), .carry(carry[898]), .a(a[898]), .b(b[898]),
.c(c[898]));
    xtfa i899(.sum(sum[899]), .carry(carry[899]), .a(a[899]), .b(b[899]),
.c(c[899]));
    xtfa i900(.sum(sum[900]), .carry(carry[900]), .a(a[900]), .b(b[900]),
.c(c[900]));
    xtfa i901(.sum(sum[901]), .carry(carry[901]), .a(a[901]), .b(b[901]),
.c(c[901]));
    xtfa i902(.sum(sum[902]), .carry(carry[902]), .a(a[902]), .b(b[902]),
.c(c[902]));
    xtfa i903(.sum(sum[903]), .carry(carry[903]), .a(a[903]), .b(b[903]),
.c(c[903]));
    xtfa i904(.sum(sum[904]), .carry(carry[904]), .a(a[904]), .b(b[904]),
.c(c[904]));
    xtfa i905(.sum(sum[905]), .carry(carry[905]), .a(a[905]), .b(b[905]),
.c(c[905]));
    xtfa i906(.sum(sum[906]), .carry(carry[906]), .a(a[906]), .b(b[906]),
.c(c[906]));
    xtfa i907(.sum(sum[907]), .carry(carry[907]), .a(a[907]), .b(b[907]),
.c(c[907]));
    xtfa i908(.sum(sum[908]), .carry(carry[908]), .a(a[908]), .b(b[908]),
.c(c[908]));
    xtfa i909(.sum(sum[909]), .carry(carry[909]), .a(a[909]), .b(b[909]),
.c(c[909]));
    xtfa i910(.sum(sum[910]), .carry(carry[910]), .a(a[910]), .b(b[910]),
.c(c[910]));
    xtfa i911(.sum(sum[911]), .carry(carry[911]), .a(a[911]), .b(b[911]),
.c(c[911]));
    xtfa i912(.sum(sum[912]), .carry(carry[912]), .a(a[912]), .b(b[912]),
.c(c[912]));
    xtfa i913(.sum(sum[913]), .carry(carry[913]), .a(a[913]), .b(b[913]),
.c(c[913]));
    xtfa i914(.sum(sum[914]), .carry(carry[914]), .a(a[914]), .b(b[914]),
.c(c[914]));
    xtfa i915(.sum(sum[915]), .carry(carry[915]), .a(a[915]), .b(b[915]),
.c(c[915]));
    xtfa i916(.sum(sum[916]), .carry(carry[916]), .a(a[916]), .b(b[916]),
.c(c[916]));
    xtfa i917(.sum(sum[917]), .carry(carry[917]), .a(a[917]), .b(b[917]),
.c(c[917]));
    xtfa i918(.sum(sum[918]), .carry(carry[918]), .a(a[918]), .b(b[918]),
.c(c[918]));
    xtfa i919(.sum(sum[919]), .carry(carry[919]), .a(a[919]), .b(b[919]),
.c(c[919]));
    xtfa i920(.sum(sum[920]), .carry(carry[920]), .a(a[920]), .b(b[920]),
.c(c[920]));
    xtfa i921(.sum(sum[921]), .carry(carry[921]), .a(a[921]), .b(b[921]),
.c(c[921]));
    xtfa i922(.sum(sum[922]), .carry(carry[922]), .a(a[922]), .b(b[922]),
.c(c[922]));
    xtfa i923(.sum(sum[923]), .carry(carry[923]), .a(a[923]), .b(b[923]),
.c(c[923]));
    xtfa i924(.sum(sum[924]), .carry(carry[924]), .a(a[924]), .b(b[924]),
.c(c[924]));
```

```
    xtfa i925(.sum(sum[925]), .carry(carry[925]), .a(a[925]), .b(b[925]),
.c(c[925]));
    xtfa i926(.sum(sum[926]), .carry(carry[926]), .a(a[926]), .b(b[926]),
.c(c[926]));
    xtfa i927(.sum(sum[927]), .carry(carry[927]), .a(a[927]), .b(b[927]),
.c(c[927]));
    xtfa i928(.sum(sum[928]), .carry(carry[928]), .a(a[928]), .b(b[928]),
.c(c[928]));
    xtfa i929(.sum(sum[929]), .carry(carry[929]), .a(a[929]), .b(b[929]),
.c(c[929]));
    xtfa i930(.sum(sum[930]), .carry(carry[930]), .a(a[930]), .b(b[930]),
.c(c[930]));
    xtfa i931(.sum(sum[931]), .carry(carry[931]), .a(a[931]), .b(b[931]),
.c(c[931]));
    xtfa i932(.sum(sum[932]), .carry(carry[932]), .a(a[932]), .b(b[932]),
.c(c[932]));
    xtfa i933(.sum(sum[933]), .carry(carry[933]), .a(a[933]), .b(b[933]),
.c(c[933]));
    xtfa i934(.sum(sum[934]), .carry(carry[934]), .a(a[934]), .b(b[934]),
.c(c[934]));
    xtfa i935(.sum(sum[935]), .carry(carry[935]), .a(a[935]), .b(b[935]),
.c(c[935]));
    xtfa i936(.sum(sum[936]), .carry(carry[936]), .a(a[936]), .b(b[936]),
.c(c[936]));
    xtfa i937(.sum(sum[937]), .carry(carry[937]), .a(a[937]), .b(b[937]),
.c(c[937]));
    xtfa i938(.sum(sum[938]), .carry(carry[938]), .a(a[938]), .b(b[938]),
.c(c[938]));
    xtfa i939(.sum(sum[939]), .carry(carry[939]), .a(a[939]), .b(b[939]),
.c(c[939]));
    xtfa i940(.sum(sum[940]), .carry(carry[940]), .a(a[940]), .b(b[940]),
.c(c[940]));
    xtfa i941(.sum(sum[941]), .carry(carry[941]), .a(a[941]), .b(b[941]),
.c(c[941]));
    xtfa i942(.sum(sum[942]), .carry(carry[942]), .a(a[942]), .b(b[942]),
.c(c[942]));
    xtfa i943(.sum(sum[943]), .carry(carry[943]), .a(a[943]), .b(b[943]),
.c(c[943]));
    xtfa i944(.sum(sum[944]), .carry(carry[944]), .a(a[944]), .b(b[944]),
.c(c[944]));
    xtfa i945(.sum(sum[945]), .carry(carry[945]), .a(a[945]), .b(b[945]),
.c(c[945]));
    xtfa i946(.sum(sum[946]), .carry(carry[946]), .a(a[946]), .b(b[946]),
.c(c[946]));
    xtfa i947(.sum(sum[947]), .carry(carry[947]), .a(a[947]), .b(b[947]),
.c(c[947]));
    xtfa i948(.sum(sum[948]), .carry(carry[948]), .a(a[948]), .b(b[948]),
.c(c[948]));
    xtfa i949(.sum(sum[949]), .carry(carry[949]), .a(a[949]), .b(b[949]),
.c(c[949]));
    xtfa i950(.sum(sum[950]), .carry(carry[950]), .a(a[950]), .b(b[950]),
.c(c[950]));
    xtfa i951(.sum(sum[951]), .carry(carry[951]), .a(a[951]), .b(b[951]),
.c(c[951]));
    xtfa i952(.sum(sum[952]), .carry(carry[952]), .a(a[952]), .b(b[952]),
.c(c[952]));
```

```
    xtfa i953(.sum(sum[953]), .carry(carry[953]), .a(a[953]), .b(b[953]),
.c(c[953]));
    xtfa i954(.sum(sum[954]), .carry(carry[954]), .a(a[954]), .b(b[954]),
.c(c[954]));
    xtfa i955(.sum(sum[955]), .carry(carry[955]), .a(a[955]), .b(b[955]),
.c(c[955]));
    xtfa i956(.sum(sum[956]), .carry(carry[956]), .a(a[956]), .b(b[956]),
.c(c[956]));
    xtfa i957(.sum(sum[957]), .carry(carry[957]), .a(a[957]), .b(b[957]),
.c(c[957]));
    xtfa i958(.sum(sum[958]), .carry(carry[958]), .a(a[958]), .b(b[958]),
.c(c[958]));
    xtfa i959(.sum(sum[959]), .carry(carry[959]), .a(a[959]), .b(b[959]),
.c(c[959]));
    xtfa i960(.sum(sum[960]), .carry(carry[960]), .a(a[960]), .b(b[960]),
.c(c[960]));
    xtfa i961(.sum(sum[961]), .carry(carry[961]), .a(a[961]), .b(b[961]),
.c(c[961]));
    xtfa i962(.sum(sum[962]), .carry(carry[962]), .a(a[962]), .b(b[962]),
.c(c[962]));
    xtfa i963(.sum(sum[963]), .carry(carry[963]), .a(a[963]), .b(b[963]),
.c(c[963]));
    xtfa i964(.sum(sum[964]), .carry(carry[964]), .a(a[964]), .b(b[964]),
.c(c[964]));
    xtfa i965(.sum(sum[965]), .carry(carry[965]), .a(a[965]), .b(b[965]),
.c(c[965]));
    xtfa i966(.sum(sum[966]), .carry(carry[966]), .a(a[966]), .b(b[966]),
.c(c[966]));
    xtfa i967(.sum(sum[967]), .carry(carry[967]), .a(a[967]), .b(b[967]),
.c(c[967]));
    xtfa i968(.sum(sum[968]), .carry(carry[968]), .a(a[968]), .b(b[968]),
.c(c[968]));
    xtfa i969(.sum(sum[969]), .carry(carry[969]), .a(a[969]), .b(b[969]),
.c(c[969]));
    xtfa i970(.sum(sum[970]), .carry(carry[970]), .a(a[970]), .b(b[970]),
.c(c[970]));
    xtfa i971(.sum(sum[971]), .carry(carry[971]), .a(a[971]), .b(b[971]),
.c(c[971]));
    xtfa i972(.sum(sum[972]), .carry(carry[972]), .a(a[972]), .b(b[972]),
.c(c[972]));
    xtfa i973(.sum(sum[973]), .carry(carry[973]), .a(a[973]), .b(b[973]),
.c(c[973]));
    xtfa i974(.sum(sum[974]), .carry(carry[974]), .a(a[974]), .b(b[974]),
.c(c[974]));
    xtfa i975(.sum(sum[975]), .carry(carry[975]), .a(a[975]), .b(b[975]),
.c(c[975]));
    xtfa i976(.sum(sum[976]), .carry(carry[976]), .a(a[976]), .b(b[976]),
.c(c[976]));
    xtfa i977(.sum(sum[977]), .carry(carry[977]), .a(a[977]), .b(b[977]),
.c(c[977]));
    xtfa i978(.sum(sum[978]), .carry(carry[978]), .a(a[978]), .b(b[978]),
.c(c[978]));
    xtfa i979(.sum(sum[979]), .carry(carry[979]), .a(a[979]), .b(b[979]),
.c(c[979]));
    xtfa i980(.sum(sum[980]), .carry(carry[980]), .a(a[980]), .b(b[980]),
.c(c[980]));
```

```
    xtfa i981(.sum(sum[981]), .carry(carry[981]), .a(a[981]), .b(b[981]),
.c(c[981]));
    xtfa i982(.sum(sum[982]), .carry(carry[982]), .a(a[982]), .b(b[982]),
.c(c[982]));
    xtfa i983(.sum(sum[983]), .carry(carry[983]), .a(a[983]), .b(b[983]),
.c(c[983]));
    xtfa i984(.sum(sum[984]), .carry(carry[984]), .a(a[984]), .b(b[984]),
.c(c[984]));
    xtfa i985(.sum(sum[985]), .carry(carry[985]), .a(a[985]), .b(b[985]),
.c(c[985]));
    xtfa i986(.sum(sum[986]), .carry(carry[986]), .a(a[986]), .b(b[986]),
.c(c[986]));
    xtfa i987(.sum(sum[987]), .carry(carry[987]), .a(a[987]), .b(b[987]),
.c(c[987]));
    xtfa i988(.sum(sum[988]), .carry(carry[988]), .a(a[988]), .b(b[988]),
.c(c[988]));
    xtfa i989(.sum(sum[989]), .carry(carry[989]), .a(a[989]), .b(b[989]),
.c(c[989]));
    xtfa i990(.sum(sum[990]), .carry(carry[990]), .a(a[990]), .b(b[990]),
.c(c[990]));
    xtfa i991(.sum(sum[991]), .carry(carry[991]), .a(a[991]), .b(b[991]),
.c(c[991]));
    xtfa i992(.sum(sum[992]), .carry(carry[992]), .a(a[992]), .b(b[992]),
.c(c[992]));
    xtfa i993(.sum(sum[993]), .carry(carry[993]), .a(a[993]), .b(b[993]),
.c(c[993]));
    xtfa i994(.sum(sum[994]), .carry(carry[994]), .a(a[994]), .b(b[994]),
.c(c[994]));
    xtfa i995(.sum(sum[995]), .carry(carry[995]), .a(a[995]), .b(b[995]),
.c(c[995]));
    xtfa i996(.sum(sum[996]), .carry(carry[996]), .a(a[996]), .b(b[996]),
.c(c[996]));
    xtfa i997(.sum(sum[997]), .carry(carry[997]), .a(a[997]), .b(b[997]),
.c(c[997]));
    xtfa i998(.sum(sum[998]), .carry(carry[998]), .a(a[998]), .b(b[998]),
.c(c[998]));
    xtfa i999(.sum(sum[999]), .carry(carry[999]), .a(a[999]), .b(b[999]),
.c(c[999]));
    xtfa i1000(.sum(sum[1000]), .carry(carry[1000]), .a(a[1000]), .b(b[1000]),
.c(c[1000]));
    xtfa i1001(.sum(sum[1001]), .carry(carry[1001]), .a(a[1001]), .b(b[1001]),
.c(c[1001]));
    xtfa i1002(.sum(sum[1002]), .carry(carry[1002]), .a(a[1002]), .b(b[1002]),
.c(c[1002]));
    xtfa i1003(.sum(sum[1003]), .carry(carry[1003]), .a(a[1003]), .b(b[1003]),
.c(c[1003]));
    xtfa i1004(.sum(sum[1004]), .carry(carry[1004]), .a(a[1004]), .b(b[1004]),
.c(c[1004]));
    xtfa i1005(.sum(sum[1005]), .carry(carry[1005]), .a(a[1005]), .b(b[1005]),
.c(c[1005]));
    xtfa i1006(.sum(sum[1006]), .carry(carry[1006]), .a(a[1006]), .b(b[1006]),
.c(c[1006]));
    xtfa i1007(.sum(sum[1007]), .carry(carry[1007]), .a(a[1007]), .b(b[1007]),
.c(c[1007]));
    xtfa i1008(.sum(sum[1008]), .carry(carry[1008]), .a(a[1008]), .b(b[1008]),
.c(c[1008]));
```

```
    xtfa i1009(.sum(sum[1009]), .carry(carry[1009]), .a(a[1009]), .b(b[1009]),
.c(c[1009]));
    xtfa i1010(.sum(sum[1010]), .carry(carry[1010]), .a(a[1010]), .b(b[1010]),
.c(c[1010]));
    xtfa i1011(.sum(sum[1011]), .carry(carry[1011]), .a(a[1011]), .b(b[1011]),
.c(c[1011]));
    xtfa i1012(.sum(sum[1012]), .carry(carry[1012]), .a(a[1012]), .b(b[1012]),
.c(c[1012]));
    xtfa i1013(.sum(sum[1013]), .carry(carry[1013]), .a(a[1013]), .b(b[1013]),
.c(c[1013]));
    xtfa i1014(.sum(sum[1014]), .carry(carry[1014]), .a(a[1014]), .b(b[1014]),
.c(c[1014]));
    xtfa i1015(.sum(sum[1015]), .carry(carry[1015]), .a(a[1015]), .b(b[1015]),
.c(c[1015]));
    xtfa i1016(.sum(sum[1016]), .carry(carry[1016]), .a(a[1016]), .b(b[1016]),
.c(c[1016]));
    xtfa i1017(.sum(sum[1017]), .carry(carry[1017]), .a(a[1017]), .b(b[1017]),
.c(c[1017]));
    xtfa i1018(.sum(sum[1018]), .carry(carry[1018]), .a(a[1018]), .b(b[1018]),
.c(c[1018]));
    xtfa i1019(.sum(sum[1019]), .carry(carry[1019]), .a(a[1019]), .b(b[1019]),
.c(c[1019]));
    xtfa i1020(.sum(sum[1020]), .carry(carry[1020]), .a(a[1020]), .b(b[1020]),
.c(c[1020]));
    xtfa i1021(.sum(sum[1021]), .carry(carry[1021]), .a(a[1021]), .b(b[1021]),
.c(c[1021]));
    xtfa i1022(.sum(sum[1022]), .carry(carry[1022]), .a(a[1022]), .b(b[1022]),
.c(c[1022]));
    xtfa i1023(.sum(sum[1023]), .carry(carry[1023]), .a(a[1023]), .b(b[1023]),
.c(c[1023]));
endmodule


// Local Variables: ***
// mode: verilog ***
// End: ***
```

verysys/verify_sem.v

```
module xmTIE_gf_Regfile(rd0_data_C1, rd0_addr_C0, rd0_width8_C0, rd0_use1_C0,
  rd1_data_C1, rd1_addr_C0, rd1_width8_C0, rd1_use1_C0, rd2_data_C1,
  rd2_addr_C0, rd2_width8_C0, rd2_use1_C0, wd_addr_C0, wd_width8_C0,
  wd_def1_C0, wd_def2_C0, wd_data8_C1, wd_data8_C2, wd_wen_C1, wd_wen_C2,
  Kill_E, KillPipe_W, Stall_R, clk);
    output [7:0] rd0_data_C1;
    input [3:0] rd0_addr_C0;
    input rd0_width8_C0;
    input rd0_use1_C0;
    output [7:0] rd1_data_C1;
    input [3:0] rd1_addr_C0;
    input rd1_width8_C0;
    input rd1_use1_C0;
    output [7:0] rd2_data_C1;
    input [3:0] rd2_addr_C0;
    input rd2_width8_C0;
    input rd2_use1_C0;
```

```
input [3:0] wd_addr_C0;
input wd_width8_C0;
input wd_def1_C0;
input wd_def2_C0;
input [7:0] wd_data8_C1;
input [7:0] wd_data8_C2;
input wd_wen_C1;
input wd_wen_C2;
input Kill_E;
input KillPipe_W;
output Stall_R;
input clk;


/*********************************************************************
      READ PORT rd0
 *********************************************************************/
// compute the address mask
wire rd0_addr_mask_C0 = 1'd0;

// masked address pipeline
wire rd0_maddr_C0 = 1'd0;

// bank-qualified use
wire rd0_use1_bank0_C0 = (rd0_use1_C0 & (rd0_maddr_C0 == (1'd0 &
rd0_addr_mask_C0)));

// alignment mux for use 1
wire [7:0] rd0_data_bank0_C1;
assign rd0_data_C1[7:0] = rd0_data_bank0_C1;


/*********************************************************************
      READ PORT rd1
 *********************************************************************/
// compute the address mask
wire rd1_addr_mask_C0 = 1'd0;

// masked address pipeline
wire rd1_maddr_C0 = 1'd0;

// bank-qualified use
wire rd1_use1_bank0_C0 = (rd1_use1_C0 & (rd1_maddr_C0 == (1'd0 &
rd1_addr_mask_C0)));

// alignment mux for use 1
wire [7:0] rd1_data_bank0_C1;
assign rd1_data_C1[7:0] = rd1_data_bank0_C1;


/*********************************************************************
      READ PORT rd2
 *********************************************************************/
// compute the address mask
wire rd2_addr_mask_C0 = 1'd0;

// masked address pipeline
```

```
    wire rd2_maddr_C0 = 1'd0;

    // bank-qualified use
    wire rd2_use1_bank0_C0 = (rd2_use1_C0 & (rd2_maddr_C0 == (1'd0 &
rd2_addr_mask_C0)));

    // alignment mux for use 1
    wire [7:0] rd2_data_bank0_C1;
    assign rd2_data_C1[7:0] = rd2_data_bank0_C1;


    /***********************************************************************
          WRITE PORT wd
     ***********************************************************************/
    // compute the address mask
    wire wd_addr_mask_C0 = 1'd0;

    // bank-qualified write def for port wd
    wire wd_def1_bank0_C0 = (wd_def1_C0 & ((wd_addr_C0 & wd_addr_mask_C0) ==
(1'd0 & wd_addr_mask_C0)));
    wire wd_def2_bank0_C0 = (wd_def2_C0 & ((wd_addr_C0 & wd_addr_mask_C0) ==
(1'd0 & wd_addr_mask_C0)));

    // write mux for def 1
    wire [7:0] wd_wdata_C1;
    assign wd_wdata_C1 = {1{wd_data8_C1[7:0]}};

    // write mux for def 2
    wire [7:0] wd_wdata_C2;
    assign wd_wdata_C2 = {1{wd_data8_C2[7:0]}};


    wire Stall_R0;
    /***********************************************************************
          PIPELINED BANK
     ***********************************************************************/
    xmTIE_gf_Regfile_bank TIE_gf_Regfile_bank0(rd0_data_bank0_C1,
        rd0_addr_C0[3:0], rd0_use1_bank0_C0, rd1_data_bank0_C1,
rd1_addr_C0[3:0],
        rd1_use1_bank0_C0, rd2_data_bank0_C1, rd2_addr_C0[3:0],
rd2_use1_bank0_C0,
        wd_addr_C0[3:0], wd_def1_bank0_C0, wd_def2_bank0_C0, wd_wdata_C1[7:0],
        wd_wdata_C2[7:0], wd_wen_C1, wd_wen_C2, Kill_E, KillPipe_W, Stall_R0,
        clk);

    assign Stall_R = Stall_R0 | 1'b0;

endmodule


module xmTIE_gf_Regfile_bank(rd0_data_C1, rd0_addr_C0, rd0_use1_C0,
  rd1_data_C1, rd1_addr_C0, rd1_use1_C0, rd2_data_C1, rd2_addr_C0,
  rd2_use1_C0, wd_addr_C0, wd_def1_C0, wd_def2_C0, wd_data_C1, wd_data_C2,
  wd_wen_C1, wd_wen_C2, Kill_E, KillPipe_W, Stall_R, clk);
    output [7:0] rd0_data_C1;
    input [3:0] rd0_addr_C0;
    input rd0_use1_C0;
```

```
output [7:0] rd1_data_C1;
input [3:0] rd1_addr_C0;
input rd1_use1_C0;
output [7:0] rd2_data_C1;
input [3:0] rd2_addr_C0;
input rd2_use1_C0;
input [3:0] wd_addr_C0;
input wd_def1_C0;
input wd_def2_C0;
input [7:0] wd_data_C1;
input [7:0] wd_data_C2;
input wd_wen_C1;
input wd_wen_C2;
input Kill_E;
input KillPipe_W;
output Stall_R;
input clk;

wire rd0_use2_C0 = 1'd0;
wire rd1_use2_C0 = 1'd0;
wire rd2_use2_C0 = 1'd0;

wire kill_C0 = KillPipe_W;
wire kill_C1 = KillPipe_W | Kill_E;
wire kill_C2 = KillPipe_W;
wire kill_C3 = KillPipe_W;

// write definition pipeline
wire wd_ns_def1_C0 = wd_def1_C0 & 1'b1 & ~kill_C0;
wire wd_def1_C1;
xtdelay1 #(1) iwd_def1_C1(wd_def1_C1, wd_ns_def1_C0, clk);
wire wd_ns_def2_C0 = wd_def2_C0 & 1'b1 & ~kill_C0;
wire wd_def2_C1;
xtdelay1 #(1) iwd_def2_C1(wd_def2_C1, wd_ns_def2_C0, clk);
wire wd_ns_def2_C1 = wd_def2_C1 & wd_wen_C1 & ~kill_C1;
wire wd_def2_C2;
xtdelay1 #(1) iwd_def2_C2(wd_def2_C2, wd_ns_def2_C1, clk);

// write enable pipeline
wire wd_we_C2;
wire wd_we_C3;
wire wd_ns_we_C1 = (1'd0 | (wd_def1_C1 & wd_wen_C1)) & ~kill_C1;
wire wd_ns_we_C2 = (wd_we_C2 | (wd_def2_C2 & wd_wen_C2)) & ~kill_C2;
wire wd_ns_we_C3 = (wd_we_C3 | (1'd0 & 1'd0)) & ~kill_C3;
xtdelay1 #(1) iwd_we_C2(wd_we_C2, wd_ns_we_C1, clk);
xtdelay1 #(1) iwd_we_C3(wd_we_C3, wd_ns_we_C2, clk);

// write address pipeline
wire [3:0] wd_addr_C1;
wire [3:0] wd_addr_C2;
wire [3:0] wd_addr_C3;
xtdelay1 #(4) iwd_addr_C1(wd_addr_C1, wd_addr_C0, clk);
xtdelay1 #(4) iwd_addr_C2(wd_addr_C2, wd_addr_C1, clk);
xtdelay1 #(4) iwd_addr_C3(wd_addr_C3, wd_addr_C2, clk);

// write data pipeline
wire [7:0] wd_result_C2;
```

```
    wire [7:0] wd_result_C3;
    wire [7:0] wd_mux_C1 = wd_data_C1;
    wire [7:0] wd_mux_C2 = wd_def2_C2 ? wd_data_C2 : wd_result_C2;
    xtdelay1 #(8) iwd_result_C2(wd_result_C2, wd_mux_C1, clk);
    xtdelay1 #(8) iwd_result_C3(wd_result_C3, wd_mux_C2, clk);

    wire [7:0] rd0_data_C0;
    wire [7:0] rd1_data_C0;
    wire [7:0] rd2_data_C0;

    xtdelay1 #(8) ird0_data_C1(rd0_data_C1, rd0_data_C0, clk);

    xtdelay1 #(8) ird1_data_C1(rd1_data_C1, rd1_data_C0, clk);

    xtdelay1 #(8) ird2_data_C1(rd2_data_C1, rd2_data_C0, clk);

    assign Stall_R =
        ((wd_addr_C1 == rd0_addr_C0) & (
            (rd0_use1_C0 & (wd_ns_def2_C1)))) |
        ((wd_addr_C1 == rd1_addr_C0) & (
            (rd1_use1_C0 & (wd_ns_def2_C1)))) |
        ((wd_addr_C1 == rd2_addr_C0) & (
            (rd2_use1_C0 & (wd_ns_def2_C1)))) |
        1'b0;

    // verification register file replacement
    wire [7:0] xwd_verify;
    xtenflop #(8) wd_verify(xwd_verify, wd_result_C3, wd_ns_we_C3, clk);
    xtflop #(8) rd0_verify(rd0_data_C0, xwd_verify, clk);
    xtflop #(8) rd1_verify(rd1_data_C0, xwd_verify, clk);
    xtflop #(8) rd2_verify(rd2_data_C0, xwd_verify, clk);
endmodule


module xmTIE_gfmod_State(ps_data_C1, ps_width8_C0, ps_use1_C0, ns_width8_C0,
  ns_def1_C0, ns_data8_C1, ns_wen_C1, Kill_E, KillPipe_W, Stall_R, clk);
    output [7:0] ps_data_C1;
    input ps_width8_C0;
    input ps_use1_C0;
    input ns_width8_C0;
    input ns_def1_C0;
    input [7:0] ns_data8_C1;
    input ns_wen_C1;
    input Kill_E;
    input KillPipe_W;
    output Stall_R;
    input clk;

    wire ps_addr_C0 = 1'd0;
    wire ns_addr_C0 = 1'd0;
    wire ns_wen_C2 = 1'd1;

    /***********************************************************************
          READ PORT ps
     ***********************************************************************/
    // compute the address mask
    wire ps_addr_mask_C0 = 1'd0;
```

```
    // masked address pipeline
    wire ps_maddr_C0 = 1'd0;

    // bank-qualified use
    wire ps_use1_bank0_C0 = (ps_use1_C0 & (ps_maddr_C0 == (1'd0 &
ps_addr_mask_C0)));

    // alignment mux for use 1
    wire [7:0] ps_data_bank0_C1;
    assign ps_data_C1[7:0] = ps_data_bank0_C1;


    /*************************************************************************
          WRITE PORT ns
     *************************************************************************/
    // compute the address mask
    wire ns_addr_mask_C0 = 1'd0;

    // bank-qualified write def for port ns
    wire ns_def1_bank0_C0 = (ns_def1_C0 & ((ns_addr_C0 & ns_addr_mask_C0) ==
(1'd0 & ns_addr_mask_C0)));

    // write mux for def 1
    wire [7:0] ns_wdata_C1;
    assign ns_wdata_C1 = {1{ns_data8_C1[7:0]}};


    wire Stall_R0;
    /*************************************************************************
          PIPELINED BANK
     *************************************************************************/
    xmTIE_gfmod_State_bank TIE_gfmod_State_bank0(ps_data_bank0_C1,
        ps_use1_bank0_C0, ns_def1_bank0_C0, ns_wdata_C1[7:0], ns_wen_C1,
        ns_wen_C2, Kill_E, KillPipe_W, Stall_R0, clk);

    assign Stall_R = Stall_R0 | 1'b0;

endmodule


module xmTIE_gfmod_State_bank(ps_data_C1, ps_use1_C0, ns_def1_C0, ns_data_C1,
  ns_wen_C1, ns_wen_C2, Kill_E, KillPipe_W, Stall_R, clk);
    output [7:0] ps_data_C1;
    input ps_use1_C0;
    input ns_def1_C0;
    input [7:0] ns_data_C1;
    input ns_wen_C1;
    input ns_wen_C2;
    input Kill_E;
    input KillPipe_W;
    output Stall_R;
    input clk;

    wire ps_addr_C0 = 1'd0;
    wire ps_use2_C0 = 1'd0;
    wire ns_addr_C0 = 1'd0;
```

```
    wire ns_def2_C0 = 1'd0;
    wire [7:0] ns_data_C2 = 0;

    wire kill_C0 = KillPipe_W;
    wire kill_C1 = KillPipe_W | Kill_E;
    wire kill_C2 = KillPipe_W;
    wire kill_C3 = KillPipe_W;

    // write definition pipeline
    wire ns_ns_def1_C0 = ns_def1_C0 & 1'b1 & ~kill_C0;
    wire ns_def1_C1;
    xtdelay1 #(1) ins_def1_C1(ns_def1_C1, ns_ns_def1_C0, clk);
    wire ns_ns_def2_C0 = 1'd0;
    wire ns_def2_C1 = 1'd0;
    wire ns_ns_def2_C1 = 1'd0;
    wire ns_def2_C2 = 1'd0;

    // write enable pipeline
    wire ns_we_C2;
    wire ns_we_C3;
    wire ns_ns_we_C1 = (1'd0 | (ns_def1_C1 & ns_wen_C1)) & ~kill_C1;
    wire ns_ns_we_C2 = (ns_we_C2 | (ns_def2_C2 & ns_wen_C2)) & ~kill_C2;
    wire ns_ns_we_C3 = (ns_we_C3 | (1'd0 & 1'd0)) & ~kill_C3;
    xtdelay1 #(1) ins_we_C2(ns_we_C2, ns_ns_we_C1, clk);
    xtdelay1 #(1) ins_we_C3(ns_we_C3, ns_ns_we_C2, clk);

    // write address pipeline
    wire  ns_addr_C1;
    wire  ns_addr_C2;
    wire  ns_addr_C3;
    assign ns_addr_C1 = 1'd0;
    assign ns_addr_C2 = 1'd0;
    assign ns_addr_C3 = 1'd0;

    // write data pipeline
    wire [7:0] ns_result_C2;
    wire [7:0] ns_result_C3;
    wire [7:0] ns_mux_C1 = ns_data_C1;
    wire [7:0] ns_mux_C2 = ns_def2_C2 ? ns_data_C2 : ns_result_C2;
    xtdelay1 #(8) ins_result_C2(ns_result_C2, ns_mux_C1, clk);
    xtdelay1 #(8) ins_result_C3(ns_result_C3, ns_mux_C2, clk);

    wire [7:0] ps_data_C0;

    xtdelay1 #(8) ips_data_C1(ps_data_C1, ps_data_C0, clk);

    assign Stall_R =
        ((ns_addr_C1 == ps_addr_C0) & (
            (ps_use1_C0 & (ns_ns_def2_C1)))) |
        1'b0;

    // verification register file replacement
    wire [7:0] xns_verify;
    xtenflop #(8) ns_verify(xns_verify, ns_result_C3, ns_ns_we_C3, clk);
    xtflop #(8) ps_verify(ps_data_C0, xns_verify, clk);
endmodule
```

```
module xmTIE_decoder (
GFADD8,
GFADD8I,
GFMULX8,
GFRWMOD8,
LGF8_I,
SGF8_I,
LGF8_IU,
SGF8_IU,
LGF8_X,
SGF8_X,
LGF8_XU,
SGF8_XU,
RUR0,
WUR0,
imm4,
imm8,
art_use,
art_def,
ars_use,
ars_def,
arr_use,
arr_def,
br_use,
br_def,
bs_use,
bs_def,
bt_use,
bt_def,
bs4_use,
bs4_def,
bs8_use,
bs8_def,
gr_use,
gr_def,
gs_use,
gs_def,
gt_use,
gt_def,
gfmod_use1,
gfmod_def1,
AR_rd0_use1,
AR_rd0_width32,
AR_rd1_use1,
AR_rd1_width32,
AR_wd_def1,
AR_wd_width32,
gf_rd0_addr,
gf_rd0_use1,
gf_rd0_width8,
gf_rd1_addr,
gf_rd1_use1,
gf_rd1_width8,
gf_rd2_addr,
gf_rd2_use1,
gf_rd2_width8,
```

```
gf_wd_addr,
gf_wd_def2,
gf_wd_def1,
gf_wd_width8,
gf1_semantic,
gf4_semantic,
gf2_semantic,
gf3_semantic,
lgf_semantic,
sgf_semantic,
RUR0_semantic,
WUR0_semantic,
load_instruction,
store_instruction,
TIE_Inst,
Inst
);
output GFADD8;
output GFADD8I;
output GFMULX8;
output GFRWMOD8;
output LGF8_I;
output SGF8_I;
output LGF8_IU;
output SGF8_IU;
output LGF8_X;
output SGF8_X;
output LGF8_XU;
output SGF8_XU;
output RUR0;
output WUR0;
output [31:0] imm4;
output [7:0] imm8;
output art_use;
output art_def;
output ars_use;
output ars_def;
output arr_use;
output arr_def;
output br_use;
output br_def;
output bs_use;
output bs_def;
output bt_use;
output bt_def;
output bs4_use;
output bs4_def;
output bs8_use;
output bs8_def;
output gr_use;
output gr_def;
output gs_use;
output gs_def;
output gt_use;
output gt_def;
output gfmod_use1;
output gfmod_def1;
```

```
output AR_rd0_use1;
output AR_rd0_width32;
output AR_rd1_use1;
output AR_rd1_width32;
output AR_wd_def1;
output AR_wd_width32;
output [3:0] gf_rd0_addr;
output gf_rd0_use1;
output gf_rd0_width8;
output [3:0] gf_rd1_addr;
output gf_rd1_use1;
output gf_rd1_width8;
output [3:0] gf_rd2_addr;
output gf_rd2_use1;
output gf_rd2_width8;
output [3:0] gf_wd_addr;
output gf_wd_def2;
output gf_wd_def1;
output gf_wd_width8;
output gf1_semantic;
output gf4_semantic;
output gf2_semantic;
output gf3_semantic;
output lgf_semantic;
output sgf_semantic;
output RUR0_semantic;
output WUR0_semantic;
output load_instruction;
output store_instruction;
output TIE_Inst;
input [23:0] Inst;

wire [3:0] op2 = {Inst[23:20]};
wire [3:0] op1 = {Inst[19:16]};
wire [3:0] op0 = {Inst[3:0]};
wire QRST = (op0==4'b0000);
wire CUST0 = (op1==4'b0110) & QRST;
assign GFADD8 = (op2==4'b0000) & CUST0;
assign GFADD8I = (op2==4'b0100) & CUST0;
assign GFMULX8 = (op2==4'b0001) & CUST0;
assign GFRWMOD8 = (op2==4'b0010) & CUST0;
wire [3:0] r = {Inst[15:12]};
wire LSCI = (op0==4'b0011);
assign LGF8_I = (r==4'b0000) & LSCI;
assign SGF8_I = (r==4'b0001) & LSCI;
assign LGF8_IU = (r==4'b0010) & LSCI;
assign SGF8_IU = (r==4'b0011) & LSCI;
wire LSCX = (op1==4'b1000) & QRST;
assign LGF8_X = (op2==4'b0000) & LSCX;
assign SGF8_X = (op2==4'b0001) & LSCX;
assign LGF8_XU = (op2==4'b0010) & LSCX;
assign SGF8_XU = (op2==4'b0011) & LSCX;
wire [3:0] s = {Inst[11:8]};
wire [3:0] t = {Inst[7:4]};
wire [7:0] st = {s,t};
wire RST3 = (op1==4'b0011) & QRST;
wire RUR = (op2==4'b1110) & RST3;
```

```
assign RUR0 = (st==8'b00000000) & RUR;
wire [7:0] sr = {r,s};
wire WUR = (op2==4'b1111) & RST3;
assign WUR0 = (sr==8'b00000000) & WUR;
assign gfmod_use1 = GFMULX8 | GFRWMOD8 | RUR0 | 1'b0;
assign gfmod_def1 = GFRWMOD8 | WUR0 | 1'b0;
assign AR_rd0_use1 = 1'b0
     | LGF8_I
     | SGF8_I
     | LGF8_IU
     | SGF8_IU
     | LGF8_X
     | SGF8_X
     | LGF8_XU
     | SGF8_XU;
assign AR_rd0_width32 = 1'b0;
assign AR_rd1_use1 = 1'b0
     | LGF8_X
     | SGF8_X
     | LGF8_XU
     | SGF8_XU
     | WUR0;
assign AR_rd1_width32 = 1'b0;
assign AR_wd_def1 = 1'b0
     | LGF8_IU
     | SGF8_IU
     | LGF8_XU
     | SGF8_XU
     | RUR0;
assign AR_wd_width32 = 1'b0;
assign gf_rd0_use1 = 1'b0
     | GFADD8
     | GFADD8I
     | GFMULX8;
assign gf_rd0_width8 = 1'b0;
assign gf_rd1_use1 = 1'b0
     | GFADD8
     | GFRWMOD8
     | SGF8_I
     | SGF8_IU;
assign gf_rd1_width8 = 1'b0;
assign gf_rd2_use1 = 1'b0
     | SGF8_X
     | SGF8_XU;
assign gf_rd2_width8 = 1'b0;
assign gf_wd_def2 = 1'b0
     | LGF8_I
     | LGF8_IU
     | LGF8_X
     | LGF8_XU;
assign gf_wd_def1 = 1'b0
     | GFADD8
     | GFADD8I
     | GFMULX8
     | GFRWMOD8;
assign gf_wd_width8 = 1'b0;
assign art_def = 1'b0;
```

```
assign art_use = LGF8_X | SGF8_X | LGF8_XU | SGF8_XU | WUR0 | 1'b0;
assign ars_def = LGF8_IU | SGF8_IU | LGF8_XU | SGF8_XU | 1'b0;
assign ars_use = LGF8_I | SGF8_I | LGF8_IU | SGF8_IU | LGF8_X | SGF8_X |
LGF8_XU | SGF8_XU | 1'b0;
assign arr_def = RUR0 | 1'b0;
assign arr_use = 1'b0;
assign br_def = 1'b0;
assign br_use = 1'b0;
assign bs_def = 1'b0;
assign bs_use = 1'b0;
assign bt_def = 1'b0;
assign bt_use = 1'b0;
assign bs4_def = 1'b0;
assign bs4_use = 1'b0;
assign bs8_def = 1'b0;
assign bs8_use = 1'b0;
assign gr_def = GFADD8 | GFADD8I | GFMULX8 | LGF8_X | LGF8_XU | 1'b0;
assign gr_use = SGF8_X | SGF8_XU | 1'b0;
assign gs_def = 1'b0;
assign gs_use = GFADD8 | GFADD8I | GFMULX8 | 1'b0;
assign gt_def = GFRWMOD8 | LGF8_I | LGF8_IU | 1'b0;
assign gt_use = GFADD8 | GFRWMOD8 | SGF8_I | SGF8_IU | 1'b0;
wire [3:0] gr_addr = r;
wire [3:0] gs_addr = s;
wire [3:0] gt_addr = t;
assign gf_wd_addr = 4'b0
     | {4{gr_def}} & gr_addr
     | {4{gt_def}} & gt_addr;
assign gf_rd0_addr = gs_addr;
assign gf_rd1_addr = gt_addr;
assign gf_rd2_addr = gr_addr;
assign gf1_semantic = GFADD8 | 1'b0;
assign gf4_semantic = GFADD8I | 1'b0;
assign gf2_semantic = GFMULX8 | 1'b0;
assign gf3_semantic = GFRWMOD8 | 1'b0;
assign lgf_semantic = LGF8_I | LGF8_IU | LGF8_X | LGF8_XU | 1'b0;
assign sgf_semantic = SGF8_I | SGF8_IU | SGF8_X | SGF8_XU | 1'b0;
assign RUR0_semantic = RUR0 | 1'b0;
assign WUR0_semantic = WUR0 | 1'b0;
assign imm4 = t;
wire [7:0] imm8 = {Inst[23:16]};
assign load_instruction = 1'b0
    | LGF8_I
    | LGF8_IU
    | LGF8_X
    | LGF8_XU;
assign store_instruction = 1'b0
    | SGF8_I
    | SGF8_IU
    | SGF8_X
    | SGF8_XU;
assign TIE_Inst = 1'b0
    | GFADD8
    | GFADD8I
    | GFMULX8
    | GFRWMOD8
    | LGF8_I
```

```
        | SGF8_I
        | LGF8_IU
        | SGF8_IU
        | LGF8_X
        | SGF8_X
        | LGF8_XU
        | SGF8_XU
        | RUR0
        | WUR0;
endmodule

module xmTIE_gf1 (
GFADD8_C0,
gr_o_C1,
gr_kill_C1,
gs_i_C1,
gt_i_C1,
clk
);
input GFADD8_C0;
output [7:0] gr_o_C1;
output gr_kill_C1;
input [7:0] gs_i_C1;
input [7:0] gt_i_C1;
input clk;
assign gr_o_C1 = (gs_i_C1) ^ (gt_i_C1);
wire GFADD8_C1;
xtdelay1 #(1) iGFADD8_C1(.xtin(GFADD8_C0), .xtout(GFADD8_C1), .clk(clk));
assign gr_kill_C1 = (1'b0) & (GFADD8_C1);
endmodule

module xmTIE_gf4 (
GFADD8I_C0,
gr_o_C1,
gr_kill_C1,
gs_i_C1,
imm4_C0,
clk
);
input GFADD8I_C0;
output [7:0] gr_o_C1;
output gr_kill_C1;
input [7:0] gs_i_C1;
input [31:0] imm4_C0;
input clk;
wire [31:0] imm4_C1;
xtdelay1 #(32) iimm4_C1(.xtin(imm4_C0), .xtout(imm4_C1), .clk(clk));
assign gr_o_C1 = (gs_i_C1) ^ (imm4_C1);
wire GFADD8I_C1;
xtdelay1 #(1) iGFADD8I_C1(.xtin(GFADD8I_C0), .xtout(GFADD8I_C1), .clk(clk));
assign gr_kill_C1 = (1'b0) & (GFADD8I_C1);
endmodule

module xmTIE_gf2 (
GFMULX8_C0,
gr_o_C1,
gr_kill_C1,
```

```
gs_i_C1,
gfmod_ps_C1,
clk
);
input GFMULX8_C0;
output [7:0] gr_o_C1;
output gr_kill_C1;
input [7:0] gs_i_C1;
input [7:0] gfmod_ps_C1;
input clk;
assign gr_o_C1 = (gs_i_C1[7]) ? (({gs_i_C1[6:0], 1'b0}) ^ (gfmod_ps_C1)) :
({gs_i_C1[6:0], 1'b0});
wire GFMULX8_C1;
xtdelay1 #(1) iGFMULX8_C1(.xtin(GFMULX8_C0), .xtout(GFMULX8_C1), .clk(clk));
assign gr_kill_C1 = (1'b0) & (GFMULX8_C1);
endmodule

module xmTIE_gf3 (
GFRWMOD8_C0,
gt_i_C1,
gt_o_C1,
gt_kill_C1,
gfmod_ps_C1,
gfmod_ns_C1,
gfmod_kill_C1,
clk
);
input GFRWMOD8_C0;
input [7:0] gt_i_C1;
output [7:0] gt_o_C1;
output gt_kill_C1;
input [7:0] gfmod_ps_C1;
output [7:0] gfmod_ns_C1;
output gfmod_kill_C1;
input clk;
wire [7:0] t1_C1;
assign t1_C1 = gt_i_C1;
wire [7:0] t2_C1;
assign t2_C1 = gfmod_ps_C1;
assign gfmod_ns_C1 = t1_C1;
assign gt_o_C1 = t2_C1;
wire GFRWMOD8_C1;
xtdelay1 #(1) iGFRWMOD8_C1(.xtin(GFRWMOD8_C0), .xtout(GFRWMOD8_C1), .clk(clk));
assign gfmod_kill_C1 = (1'b0) & (GFRWMOD8_C1);
assign gt_kill_C1 = (1'b0) & (GFRWMOD8_C1);
endmodule

module xmTIE_lgf (
LGF8_I_C0,
LGF8_IU_C0,
LGF8_X_C0,
LGF8_XU_C0,
gt_o_C2,
gt_kill_C2,
ars_i_C1,
ars_o_C1,
ars_kill_C1,
```

```
imm8_C0,
gr_o_C2,
gr_kill_C2,
art_i_C1,
MemDataIn8_C2,
VAddrIn_C1,
LSSize_C0,
VAddrBase_C1,
VAddrIndex_C1,
VAddrOffset_C0,
LSIndexed_C0,
clk
);
input LGF8_I_C0;
input LGF8_IU_C0;
input LGF8_X_C0;
input LGF8_XU_C0;
output [7:0] gt_o_C2;
output gt_kill_C2;
input [31:0] ars_i_C1;
output [31:0] ars_o_C1;
output ars_kill_C1;
input [7:0] imm8_C0;
output [7:0] gr_o_C2;
output gr_kill_C2;
input [31:0] art_i_C1;
input [7:0] MemDataIn8_C2;
input [31:0] VAddrIn_C1;
output [4:0] LSSize_C0;
output [31:0] VAddrBase_C1;
output [31:0] VAddrIndex_C1;
output [31:0] VAddrOffset_C0;
output LSIndexed_C0;
input clk;
wire indexed_C0;
assign indexed_C0 = (LGF8_X_C0) | (LGF8_XU_C0);
assign LSSize_C0 = 32'h1;
assign VAddrBase_C1 = ars_i_C1;
assign LSIndexed_C0 = indexed_C0;
assign VAddrOffset_C0 = imm8_C0;
assign VAddrIndex_C1 = art_i_C1;
assign gt_o_C2 = MemDataIn8_C2;
assign gr_o_C2 = MemDataIn8_C2;
assign ars_o_C1 = VAddrIn_C1;
wire LGF8_I_C2;
xtdelay2 #(1) iLGF8_I_C2(.xtin(LGF8_I_C0), .xtout(LGF8_I_C2), .clk(clk));
wire LGF8_IU_C2;
xtdelay2 #(1) iLGF8_IU_C2(.xtin(LGF8_IU_C0), .xtout(LGF8_IU_C2), .clk(clk));
assign gt_kill_C2 = (1'b0) & ((LGF8_I_C2) | (LGF8_IU_C2));
wire LGF8_IU_C1;
xtdelay1 #(1) iLGF8_IU_C1(.xtin(LGF8_IU_C0), .xtout(LGF8_IU_C1), .clk(clk));
wire LGF8_XU_C1;
xtdelay1 #(1) iLGF8_XU_C1(.xtin(LGF8_XU_C0), .xtout(LGF8_XU_C1), .clk(clk));
assign ars_kill_C1 = (1'b0) & ((LGF8_IU_C1) | (LGF8_XU_C1));
wire LGF8_X_C2;
xtdelay2 #(1) iLGF8_X_C2(.xtin(LGF8_X_C0), .xtout(LGF8_X_C2), .clk(clk));
wire LGF8_XU_C2;
```

```
xtdelay2 #(1) iLGF8_XU_C2(.xtin(LGF8_XU_C0), .xtout(LGF8_XU_C2), .clk(clk));
assign gr_kill_C2 = (1'b0) & ((LGF8_X_C2) | (LGF8_XU_C2));
endmodule

module xmTIE_sgf (
SGF8_I_C0,
SGF8_IU_C0,
SGF8_X_C0,
SGF8_XU_C0,
gt_i_C1,
ars_i_C1,
ars_o_C1,
ars_kill_C1,
imm8_C0,
gr_i_C1,
art_i_C1,
VAddrIn_C1,
LSSize_C0,
MemDataOut8_C1,
VAddrBase_C1,
VAddrIndex_C1,
VAddrOffset_C0,
LSIndexed_C0,
clk
);
input SGF8_I_C0;
input SGF8_IU_C0;
input SGF8_X_C0;
input SGF8_XU_C0;
input [7:0] gt_i_C1;
input [31:0] ars_i_C1;
output [31:0] ars_o_C1;
output ars_kill_C1;
input [7:0] imm8_C0;
input [7:0] gr_i_C1;
input [31:0] art_i_C1;
input [31:0] VAddrIn_C1;
output [4:0] LSSize_C0;
output [7:0] MemDataOut8_C1;
output [31:0] VAddrBase_C1;
output [31:0] VAddrIndex_C1;
output [31:0] VAddrOffset_C0;
output LSIndexed_C0;
input clk;
wire indexed_C0;
assign indexed_C0 = (SGF8_X_C0) | (SGF8_XU_C0);
assign LSSize_C0 = 32'h1;
assign VAddrBase_C1 = ars_i_C1;
assign LSIndexed_C0 = indexed_C0;
assign VAddrOffset_C0 = imm8_C0;
assign VAddrIndex_C1 = art_i_C1;
wire SGF8_X_C1;
xtdelay1 #(1) iSGF8_X_C1(.xtin(SGF8_X_C0), .xtout(SGF8_X_C1), .clk(clk));
wire SGF8_XU_C1;
xtdelay1 #(1) iSGF8_XU_C1(.xtin(SGF8_XU_C0), .xtout(SGF8_XU_C1), .clk(clk));
assign MemDataOut8_C1 = ((SGF8_X_C1) | (SGF8_XU_C1)) ? (gr_i_C1) : (gt_i_C1);
assign ars_o_C1 = VAddrIn_C1;
```

```
wire SGF8_IU_C1;
xtdelay1 #(1) iSGF8_IU_C1(.xtin(SGF8_IU_C0), .xtout(SGF8_IU_C1), .clk(clk));
assign ars_kill_C1 = (1'b0) & ((SGF8_IU_C1) | (SGF8_XU_C1));
endmodule

module xmTIE_RUR0 (
RUR0_C0,
arr_o_C1,
arr_kill_C1,
gfmod_ps_C1,
clk
);
input RUR0_C0;
output [31:0] arr_o_C1;
output arr_kill_C1;
input [7:0] gfmod_ps_C1;
input clk;
assign arr_o_C1 = {gfmod_ps_C1};
wire RUR0_C1;
xtdelay1 #(1) iRUR0_C1(.xtin(RUR0_C0), .xtout(RUR0_C1), .clk(clk));
assign arr_kill_C1 = (1'b0) & (RUR0_C1);
endmodule

module xmTIE_WUR0 (
WUR0_C0,
art_i_C1,
gfmod_ns_C1,
gfmod_kill_C1,
clk
);
input WUR0_C0;
input [31:0] art_i_C1;
output [7:0] gfmod_ns_C1;
output gfmod_kill_C1;
input clk;
assign gfmod_ns_C1 = {art_i_C1[7:0]};
wire WUR0_C1;
xtdelay1 #(1) iWUR0_C1(.xtin(WUR0_C0), .xtout(WUR0_C1), .clk(clk));
assign gfmod_kill_C1 = (1'b0) & (WUR0_C1);
endmodule

module xmTIE (
TIE_inst_R,
TIE_asRead_R,
TIE_atRead_R,
TIE_atWrite_R,
TIE_arWrite_R,
TIE_asWrite_R,
TIE_aWriteM_R,
TIE_aDataKill_E,
TIE_aWriteData_E,
TIE_aDataKill_M,
TIE_aWriteData_M,
TIE_Load_R,
TIE_Store_R,
TIE_LSSize_R,
TIE_LSIndexed_R,
```

```
     TIE_LSOffset_R,
     TIE_MemLoadData_M,
     TIE_MemStoreData8_E,
     TIE_MemStoreData16_E,
     TIE_MemStoreData32_E,
     TIE_MemStoreData64_E,
     TIE_MemStoreData128_E,
     TIE_Stall_R,
     TIE_Exception_E,
     TIE_ExcCause_E,
     TIE_bsRead_R,
     TIE_btRead_R,
     TIE_btWrite_R,
     TIE_brWrite_R,
     TIE_bsWrite_R,
     TIE_bsReadSize_R,
     TIE_btReadSize_R,
     TIE_bWriteSize_R,
     TIE_bsReadData_E,
     TIE_btReadData_E,
     TIE_bWriteData1_E,
     TIE_bWriteData2_E,
     TIE_bWriteData4_E,
     TIE_bWriteData8_E,
     TIE_bWriteData16_E,
     TIE_bDataKill_E,
     CPEnable,
     Instr_R,
     SBus_E,
     TBus_E,
     MemOpAddr_E,
     Kill_E,
     Except_W,
     Replay_W,
     G1WCLK,
     Reset
     );
     output TIE_inst_R;
     output TIE_asRead_R;
     output TIE_atRead_R;
     output TIE_atWrite_R;
     output TIE_arWrite_R;
     output TIE_asWrite_R;
     output TIE_aWriteM_R;
     output TIE_aDataKill_E;
     output [31:0] TIE_aWriteData_E;
     output TIE_aDataKill_M;
     output [31:0] TIE_aWriteData_M;
     output TIE_Load_R;
     output TIE_Store_R;
     output [4:0] TIE_LSSize_R;
     output TIE_LSIndexed_R;
     output [31:0] TIE_LSOffset_R;
     input [127:0] TIE_MemLoadData_M;
     output [7:0] TIE_MemStoreData8_E;
     output [15:0] TIE_MemStoreData16_E;
     output [31:0] TIE_MemStoreData32_E;
```

```
output [63:0] TIE_MemStoreData64_E;
output [127:0] TIE_MemStoreData128_E;
output TIE_Stall_R;
output TIE_Exception_E;
output [5:0] TIE_ExcCause_E;
output TIE_bsRead_R;
output TIE_btRead_R;
output TIE_btWrite_R;
output TIE_brWrite_R;
output TIE_bsWrite_R;
output [4:0] TIE_bsReadSize_R;
output [4:0] TIE_btReadSize_R;
output [4:0] TIE_bWriteSize_R;
input [15:0] TIE_bsReadData_E;
input [15:0] TIE_btReadData_E;
output TIE_bWriteData1_E;
output [1:0] TIE_bWriteData2_E;
output [3:0] TIE_bWriteData4_E;
output [7:0] TIE_bWriteData8_E;
output [15:0] TIE_bWriteData16_E;
output TIE_bDataKill_E;
input [7:0] CPEnable;
input [23:0] Instr_R;
input [31:0] SBus_E;
input [31:0] TBus_E;
input [31:0] MemOpAddr_E;
input Kill_E;
input Except_W;
input Replay_W;
input G1WCLK;
input Reset;

// unused signals
wire TMode = 0;

// control signals
wire KillPipe_W;
wire clk;

// decoded signals
wire GFADD8_C0;
wire GFADD8I_C0;
wire GFMULX8_C0;
wire GFRWMOD8_C0;
wire LGF8_I_C0;
wire SGF8_I_C0;
wire LGF8_IU_C0;
wire SGF8_IU_C0;
wire LGF8_X_C0;
wire SGF8_X_C0;
wire LGF8_XU_C0;
wire SGF8_XU_C0;
wire RUR0_C0;
wire WUR0_C0;
wire [31:0] imm4_C0;
wire [7:0] imm8_C0;
wire art_use_C0;
```

```
wire art_def_C0;
wire ars_use_C0;
wire ars_def_C0;
wire arr_use_C0;
wire arr_def_C0;
wire br_use_C0;
wire br_def_C0;
wire bs_use_C0;
wire bs_def_C0;
wire bt_use_C0;
wire bt_def_C0;
wire bs4_use_C0;
wire bs4_def_C0;
wire bs8_use_C0;
wire bs8_def_C0;
wire gr_use_C0;
wire gr_def_C0;
wire gs_use_C0;
wire gs_def_C0;
wire gt_use_C0;
wire gt_def_C0;
wire gfmod_use1_C0;
wire gfmod_def1_C0;
wire AR_rd0_use1_C0;
wire AR_rd0_width32_C0;
wire AR_rd1_use1_C0;
wire AR_rd1_width32_C0;
wire AR_wd_def1_C0;
wire AR_wd_width32_C0;
wire [3:0] gf_rd0_addr_C0;
wire gf_rd0_use1_C0;
wire gf_rd0_width8_C0;
wire [3:0] gf_rd1_addr_C0;
wire gf_rd1_use1_C0;
wire gf_rd1_width8_C0;
wire [3:0] gf_rd2_addr_C0;
wire gf_rd2_use1_C0;
wire gf_rd2_width8_C0;
wire [3:0] gf_wd_addr_C0;
wire gf_wd_def2_C0;
wire gf_wd_def1_C0;
wire gf_wd_width8_C0;
wire gf1_semantic_C0;
wire gf4_semantic_C0;
wire gf2_semantic_C0;
wire gf3_semantic_C0;
wire lgf_semantic_C0;
wire sgf_semantic_C0;
wire RUR0_semantic_C0;
wire WUR0_semantic_C0;
wire load_instruction_C0;
wire store_instruction_C0;
wire TIE_Inst_C0;
wire [23:0] Inst_C0;

// state data, write-enable and stall signals
wire [7:0] gfmod_ps_C1;
```

```
wire [7:0] gfmod_ns_C1;
wire gfmod_kill_C1;
wire gfmod_Stall_C1;

// register data, write-enable and stall signals
wire [31:0] AR_rd0_data_C1;
wire [31:0] AR_rd1_data_C1;
wire [31:0] AR_wd_data32_C1;
wire AR_wd_kill_C1;
wire [7:0] gf_rd0_data_C1;
wire [7:0] gf_rd1_data_C1;
wire [7:0] gf_rd2_data_C1;
wire [7:0] gf_wd_data8_C2;
wire gf_wd_kill_C2;
wire [7:0] gf_wd_data8_C1;
wire gf_wd_kill_C1;
wire gf_Stall_C1;

// operands
wire [31:0] art_i_C1;
wire [31:0] art_o_C1;
wire art_kill_C1;
wire [31:0] ars_i_C1;
wire [31:0] ars_o_C1;
wire ars_kill_C1;
wire [31:0] arr_o_C1;
wire arr_kill_C1;
wire [7:0] gr_i_C1;
wire [7:0] gr_o_C2;
wire gr_kill_C2;
wire [7:0] gr_o_C1;
wire gr_kill_C1;
wire [7:0] gs_i_C1;
wire [7:0] gt_i_C1;
wire [7:0] gt_o_C2;
wire gt_kill_C2;
wire [7:0] gt_o_C1;
wire gt_kill_C1;

// output state of semantic gf1

// output interface of semantic gf1

// output operand of semantic gf1
wire [7:0] gf1_gr_o_C1;
wire gf1_gr_kill_C1;

// output state of semantic gf4

// output interface of semantic gf4

// output operand of semantic gf4
wire [7:0] gf4_gr_o_C1;
wire gf4_gr_kill_C1;

// output state of semantic gf2
```

```
// output interface of semantic gf2

// output operand of semantic gf2
wire [7:0] gf2_gr_o_C1;
wire gf2_gr_kill_C1;

// output state of semantic gf3
wire [7:0] gf3_gfmod_ns_C1;
wire gf3_gfmod_kill_C1;

// output interface of semantic gf3

// output operand of semantic gf3
wire [7:0] gf3_gt_o_C1;
wire gf3_gt_kill_C1;

// output state of semantic lgf

// output interface of semantic lgf
wire [4:0] lgf_LSSize_C0;
wire [31:0] lgf_VAddrBase_C1;
wire [31:0] lgf_VAddrIndex_C1;
wire [31:0] lgf_VAddrOffset_C0;
wire lgf_LSIndexed_C0;

// output operand of semantic lgf
wire [7:0] lgf_gt_o_C2;
wire lgf_gt_kill_C2;
wire [31:0] lgf_ars_o_C1;
wire lgf_ars_kill_C1;
wire [7:0] lgf_gr_o_C2;
wire lgf_gr_kill_C2;

// output state of semantic sgf

// output interface of semantic sgf
wire [4:0] sgf_LSSize_C0;
wire [7:0] sgf_MemDataOut8_C1;
wire [31:0] sgf_VAddrBase_C1;
wire [31:0] sgf_VAddrIndex_C1;
wire [31:0] sgf_VAddrOffset_C0;
wire sgf_LSIndexed_C0;

// output operand of semantic sgf
wire [31:0] sgf_ars_o_C1;
wire sgf_ars_kill_C1;

// output state of semantic RUR0

// output interface of semantic RUR0

// output operand of semantic RUR0
wire [31:0] RUR0_arr_o_C1;
wire RUR0_arr_kill_C1;

// output state of semantic WUR0
wire [7:0] WUR0_gfmod_ns_C1;
```

```
wire WUR0_gfmod_kill_C1;

// output interface of semantic WUR0

// output operand of semantic WUR0

// TIE-defined interface signals
wire [31:0] VAddr_C1;
wire [31:0] VAddrBase_C1;
wire [31:0] VAddrOffset_C0;
wire [31:0] VAddrIndex_C1;
wire [31:0] VAddrIn_C1;
wire [4:0] LSSize_C0;
wire LSIndexed_C0;
wire [127:0] MemDataIn128_C2;
wire [63:0] MemDataIn64_C2;
wire [31:0] MemDataIn32_C2;
wire [15:0] MemDataIn16_C2;
wire [7:0] MemDataIn8_C2;
wire [127:0] MemDataOut128_C1;
wire [63:0] MemDataOut64_C1;
wire [31:0] MemDataOut32_C1;
wire [15:0] MemDataOut16_C1;
wire [7:0] MemDataOut8_C1;
wire Exception_C1;
wire [5:0] ExcCause_C1;
wire [7:0] CPEnable_C1;
    xtflop #(1) reset(localReset, Reset, G1WCLK);

xmTIE_decoder TIE_decoder (
    .GFADD8(GFADD8_C0),
    .GFADD8I(GFADD8I_C0),
    .GFMULX8(GFMULX8_C0),
    .GFRWMOD8(GFRWMOD8_C0),
    .LGF8_I(LGF8_I_C0),
    .SGF8_I(SGF8_I_C0),
    .LGF8_IU(LGF8_IU_C0),
    .SGF8_IU(SGF8_IU_C0),
    .LGF8_X(LGF8_X_C0),
    .SGF8_X(SGF8_X_C0),
    .LGF8_XU(LGF8_XU_C0),
    .SGF8_XU(SGF8_XU_C0),
    .RUR0(RUR0_C0),
    .WUR0(WUR0_C0),
    .imm4(imm4_C0),
    .imm8(imm8_C0),
    .art_use(art_use_C0),
    .art_def(art_def_C0),
    .ars_use(ars_use_C0),
    .ars_def(ars_def_C0),
    .arr_use(arr_use_C0),
    .arr_def(arr_def_C0),
    .br_use(br_use_C0),
    .br_def(br_def_C0),
    .bs_use(bs_use_C0),
    .bs_def(bs_def_C0),
    .bt_use(bt_use_C0),
```

```
        .bt_def(bt_def_C0),
        .bs4_use(bs4_use_C0),
        .bs4_def(bs4_def_C0),
        .bs8_use(bs8_use_C0),
        .bs8_def(bs8_def_C0),
        .gr_use(gr_use_C0),
        .gr_def(gr_def_C0),
        .gs_use(gs_use_C0),
        .gs_def(gs_def_C0),
        .gt_use(gt_use_C0),
        .gt_def(gt_def_C0),
        .gfmod_use1(gfmod_use1_C0),
        .gfmod_def1(gfmod_def1_C0),
        .AR_rd0_use1(AR_rd0_use1_C0),
        .AR_rd0_width32(AR_rd0_width32_C0),
        .AR_rd1_use1(AR_rd1_use1_C0),
        .AR_rd1_width32(AR_rd1_width32_C0),
        .AR_wd_def1(AR_wd_def1_C0),
        .AR_wd_width32(AR_wd_width32_C0),
        .gf_rd0_addr(gf_rd0_addr_C0),
        .gf_rd0_use1(gf_rd0_use1_C0),
        .gf_rd0_width8(gf_rd0_width8_C0),
        .gf_rd1_addr(gf_rd1_addr_C0),
        .gf_rd1_use1(gf_rd1_use1_C0),
        .gf_rd1_width8(gf_rd1_width8_C0),
        .gf_rd2_addr(gf_rd2_addr_C0),
        .gf_rd2_use1(gf_rd2_use1_C0),
        .gf_rd2_width8(gf_rd2_width8_C0),
        .gf_wd_addr(gf_wd_addr_C0),
        .gf_wd_def2(gf_wd_def2_C0),
        .gf_wd_def1(gf_wd_def1_C0),
        .gf_wd_width8(gf_wd_width8_C0),
        .gf1_semantic(gf1_semantic_C0),
        .gf4_semantic(gf4_semantic_C0),
        .gf2_semantic(gf2_semantic_C0),
        .gf3_semantic(gf3_semantic_C0),
        .lgf_semantic(lgf_semantic_C0),
        .sgf_semantic(sgf_semantic_C0),
        .RUR0_semantic(RUR0_semantic_C0),
        .WUR0_semantic(WUR0_semantic_C0),
        .load_instruction(load_instruction_C0),
        .store_instruction(store_instruction_C0),
        .TIE_Inst(TIE_Inst_C0),
        .Inst(Inst_C0)
);

xmTIE_gf1 TIE_gf1(
        .GFADD8_C0(GFADD8_C0),
        .gr_o_C1(gf1_gr_o_C1),
        .gr_kill_C1(gf1_gr_kill_C1),
        .gs_i_C1(gs_i_C1),
        .gt_i_C1(gt_i_C1),
        .clk(clk));

xmTIE_gf4 TIE_gf4(
        .GFADD8I_C0(GFADD8I_C0),
        .gr_o_C1(gf4_gr_o_C1),
```

```
    .gr_kill_C1(gf4_gr_kill_C1),
    .gs_i_C1(gs_i_C1),
    .imm4_C0(imm4_C0),
    .clk(clk));

xmTIE_gf2 TIE_gf2(
    .GFMULX8_C0(GFMULX8_C0),
    .gr_o_C1(gf2_gr_o_C1),
    .gr_kill_C1(gf2_gr_kill_C1),
    .gs_i_C1(gs_i_C1),
    .gfmod_ps_C1(gfmod_ps_C1),
    .clk(clk));

xmTIE_gf3 TIE_gf3(
    .GFRWMOD8_C0(GFRWMOD8_C0),
    .gt_i_C1(gt_i_C1),
    .gt_o_C1(gf3_gt_o_C1),
    .gt_kill_C1(gf3_gt_kill_C1),
    .gfmod_ps_C1(gfmod_ps_C1),
    .gfmod_ns_C1(gf3_gfmod_ns_C1),
    .gfmod_kill_C1(gf3_gfmod_kill_C1),
    .clk(clk));

xmTIE_lgf TIE_lgf(
    .LGF8_I_C0(LGF8_I_C0),
    .LGF8_IU_C0(LGF8_IU_C0),
    .LGF8_X_C0(LGF8_X_C0),
    .LGF8_XU_C0(LGF8_XU_C0),
    .gt_o_C2(lgf_gt_o_C2),
    .gt_kill_C2(lgf_gt_kill_C2),
    .ars_i_C1(ars_i_C1),
    .ars_o_C1(lgf_ars_o_C1),
    .ars_kill_C1(lgf_ars_kill_C1),
    .imm8_C0(imm8_C0),
    .gr_o_C2(lgf_gr_o_C2),
    .gr_kill_C2(lgf_gr_kill_C2),
    .art_i_C1(art_i_C1),
    .MemDataIn8_C2(MemDataIn8_C2),
    .VAddrIn_C1(VAddrIn_C1),
    .LSSize_C0(lgf_LSSize_C0),
    .VAddrBase_C1(lgf_VAddrBase_C1),
    .VAddrIndex_C1(lgf_VAddrIndex_C1),
    .VAddrOffset_C0(lgf_VAddrOffset_C0),
    .LSIndexed_C0(lgf_LSIndexed_C0),
    .clk(clk));

xmTIE_sgf TIE_sgf(
    .SGF8_I_C0(SGF8_I_C0),
    .SGF8_IU_C0(SGF8_IU_C0),
    .SGF8_X_C0(SGF8_X_C0),
    .SGF8_XU_C0(SGF8_XU_C0),
    .gt_i_C1(gt_i_C1),
    .ars_i_C1(ars_i_C1),
    .ars_o_C1(sgf_ars_o_C1),
    .ars_kill_C1(sgf_ars_kill_C1),
    .imm8_C0(imm8_C0),
    .gr_i_C1(gr_i_C1),
```

```
        .art_i_C1(art_i_C1),
        .VAddrIn_C1(VAddrIn_C1),
        .LSSize_C0(sgf_LSSize_C0),
        .MemDataOut8_C1(sgf_MemDataOut8_C1),
        .VAddrBase_C1(sgf_VAddrBase_C1),
        .VAddrIndex_C1(sgf_VAddrIndex_C1),
        .VAddrOffset_C0(sgf_VAddrOffset_C0),
        .LSIndexed_C0(sgf_LSIndexed_C0),
        .clk(clk));

    xmTIE_RUR0 TIE_RUR0(
        .RUR0_C0(RUR0_C0),
        .arr_o_C1(RUR0_arr_o_C1),
        .arr_kill_C1(RUR0_arr_kill_C1),
        .gfmod_ps_C1(gfmod_ps_C1),
        .clk(clk));

    xmTIE_WUR0 TIE_WUR0(
        .WUR0_C0(WUR0_C0),
        .art_i_C1(art_i_C1),
        .gfmod_ns_C1(WUR0_gfmod_ns_C1),
        .gfmod_kill_C1(WUR0_gfmod_kill_C1),
        .clk(clk));

    xmTIE_gfmod_State TIE_gfmod_State (
        .ps_width8_C0(1'b1),
        .ps_use1_C0(gfmod_use1_C0),
        .ps_data_C1(gfmod_ps_C1),
        .ns_width8_C0(1'b1),
        .ns_def1_C0(gfmod_def1_C0),
        .ns_data8_C1(gfmod_ns_C1),
        .ns_wen_C1(~gfmod_kill_C1),
        .Kill_E(Kill_E),
        .KillPipe_W(KillPipe_W),
        .Stall_R(gfmod_Stall_C1),
        .clk(clk)
    );

    xmTIE_gf_Regfile TIE_gf_Regfile (
        .rd0_addr_C0(gf_rd0_addr_C0),
        .rd0_use1_C0(gf_rd0_use1_C0),
        .rd0_data_C1(gf_rd0_data_C1),
        .rd0_width8_C0(gf_rd0_width8_C0),
        .rd1_addr_C0(gf_rd1_addr_C0),
        .rd1_use1_C0(gf_rd1_use1_C0),
        .rd1_data_C1(gf_rd1_data_C1),
        .rd1_width8_C0(gf_rd1_width8_C0),
        .rd2_addr_C0(gf_rd2_addr_C0),
        .rd2_use1_C0(gf_rd2_use1_C0),
        .rd2_data_C1(gf_rd2_data_C1),
        .rd2_width8_C0(gf_rd2_width8_C0),
        .wd_addr_C0(gf_wd_addr_C0),
        .wd_def2_C0(gf_wd_def2_C0),
        .wd_wen_C2(~gf_wd_kill_C2),
        .wd_data8_C2(gf_wd_data8_C2),
        .wd_def1_C0(gf_wd_def1_C0),
        .wd_wen_C1(~gf_wd_kill_C1),
```

```
    .wd_data8_C1(gf_wd_data8_C1),
    .wd_width8_C0(gf_wd_width8_C0),
    .Kill_E(Kill_E),
    .KillPipe_W(KillPipe_W),
    .Stall_R(gf_Stall_C1),
    .clk(clk)
);

// Stall logic
assign TIE_Stall_R = 1'b0
          | gf_Stall_C1
          | gfmod_Stall_C1;

// pipeline semantic select signals to each stage
wire lgf_semantic_C1;
xtdelay1 #(1) ilgf_semantic_C1(.xtin(lgf_semantic_C0), .xtout(lgf_semantic_C1),
.clk(clk));
wire sgf_semantic_C1;
xtdelay1 #(1) isgf_semantic_C1(.xtin(sgf_semantic_C0), .xtout(sgf_semantic_C1),
.clk(clk));
wire gf3_semantic_C1;
xtdelay1 #(1) igf3_semantic_C1(.xtin(gf3_semantic_C0), .xtout(gf3_semantic_C1),
.clk(clk));
wire WUR0_semantic_C1;
xtdelay1 #(1) iWUR0_semantic_C1(.xtin(WUR0_semantic_C0),
.xtout(WUR0_semantic_C1), .clk(clk));
wire RUR0_semantic_C1;
xtdelay1 #(1) iRUR0_semantic_C1(.xtin(RUR0_semantic_C0),
.xtout(RUR0_semantic_C1), .clk(clk));
wire lgf_semantic_C2;
xtdelay2 #(1) ilgf_semantic_C2(.xtin(lgf_semantic_C0), .xtout(lgf_semantic_C2),
.clk(clk));
wire gf1_semantic_C1;
xtdelay1 #(1) igf1_semantic_C1(.xtin(gf1_semantic_C0), .xtout(gf1_semantic_C1),
.clk(clk));
wire gf4_semantic_C1;
xtdelay1 #(1) igf4_semantic_C1(.xtin(gf4_semantic_C0), .xtout(gf4_semantic_C1),
.clk(clk));
wire gf2_semantic_C1;
xtdelay1 #(1) igf2_semantic_C1(.xtin(gf2_semantic_C0), .xtout(gf2_semantic_C1),
.clk(clk));

// combine output interface signals from all semantics
assign VAddr_C1 = 32'b0;
assign VAddrBase_C1 = 32'b0
          | (lgf_VAddrBase_C1 & {32{lgf_semantic_C1}})
          | (sgf_VAddrBase_C1 & {32{sgf_semantic_C1}});
assign VAddrOffset_C0 = 32'b0
          | (lgf_VAddrOffset_C0 & {32{lgf_semantic_C0}})
          | (sgf_VAddrOffset_C0 & {32{sgf_semantic_C0}});
assign VAddrIndex_C1 = 32'b0
          | (lgf_VAddrIndex_C1 & {32{lgf_semantic_C1}})
          | (sgf_VAddrIndex_C1 & {32{sgf_semantic_C1}});
assign LSSize_C0 = 5'b0
          | (lgf_LSSize_C0 & {5{lgf_semantic_C0}})
          | (sgf_LSSize_C0 & {5{sgf_semantic_C0}});
assign LSIndexed_C0 = 1'b0
```

```
            | (lgf_LSIndexed_C0 & lgf_semantic_C0)
            | (sgf_LSIndexed_C0 & sgf_semantic_C0);
assign MemDataOut128_C1 = 128'b0;
assign MemDataOut64_C1 = 64'b0;
assign MemDataOut32_C1 = 32'b0;
assign MemDataOut16_C1 = 16'b0;
assign MemDataOut8_C1 = 8'b0
            | (sgf_MemDataOut8_C1 & {8{sgf_semantic_C1}});
assign Exception_C1 = 1'b0;
assign ExcCause_C1 = 6'b0;

// combine output state signals from all semantics
assign gfmod_ns_C1 = 8'b0
            | (gf3_gfmod_ns_C1 & {8{gf3_semantic_C1}})
            | (WUR0_gfmod_ns_C1 & {8{WUR0_semantic_C1}});
assign gfmod_kill_C1 = 1'b0
            | (gf3_gfmod_kill_C1 & gf3_semantic_C1)
            | (WUR0_gfmod_kill_C1 & WUR0_semantic_C1);

// combine output operand signals from all semantics
assign art_o_C1 = 32'b0;
assign art_kill_C1 = 1'b0;
assign ars_o_C1 = 32'b0
            | (lgf_ars_o_C1 & {32{lgf_semantic_C1}})
            | (sgf_ars_o_C1 & {32{sgf_semantic_C1}});
assign ars_kill_C1 = 1'b0
            | (lgf_ars_kill_C1 & lgf_semantic_C1)
            | (sgf_ars_kill_C1 & sgf_semantic_C1);
assign arr_o_C1 = 32'b0
            | (RUR0_arr_o_C1 & {32{RUR0_semantic_C1}});
assign arr_kill_C1 = 1'b0
            | (RUR0_arr_kill_C1 & RUR0_semantic_C1);
assign gr_o_C2 = 8'b0
            | (lgf_gr_o_C2 & {8{lgf_semantic_C2}});
assign gr_kill_C2 = 1'b0
            | (lgf_gr_kill_C2 & lgf_semantic_C2);
assign gr_o_C1 = 8'b0
            | (gf1_gr_o_C1 & {8{gf1_semantic_C1}})
            | (gf4_gr_o_C1 & {8{gf4_semantic_C1}})
            | (gf2_gr_o_C1 & {8{gf2_semantic_C1}});
assign gr_kill_C1 = 1'b0
            | (gf1_gr_kill_C1 & gf1_semantic_C1)
            | (gf4_gr_kill_C1 & gf4_semantic_C1)
            | (gf2_gr_kill_C1 & gf2_semantic_C1);
assign gt_o_C2 = 8'b0
            | (lgf_gt_o_C2 & {8{lgf_semantic_C2}});
assign gt_kill_C2 = 1'b0
            | (lgf_gt_kill_C2 & lgf_semantic_C2);
assign gt_o_C1 = 8'b0
            | (gf3_gt_o_C1 & {8{gf3_semantic_C1}});
assign gt_kill_C1 = 1'b0
            | (gf3_gt_kill_C1 & gf3_semantic_C1);

// output operand to write port mapping logic
assign AR_wd_data32_C1 = ars_o_C1 | arr_o_C1 | 32'b0;
assign AR_wd_kill_C1 = ars_kill_C1 | arr_kill_C1 | 1'b0;
assign gf_wd_data8_C2 = gt_o_C2 | gr_o_C2 | 8'b0;
```

```
assign gf_wd_kill_C2 = gt_kill_C2 | gr_kill_C2 | 1'b0;
assign gf_wd_data8_C1 = gr_o_C1 | gt_o_C1 | 8'b0;
assign gf_wd_kill_C1 = gr_kill_C1 | gt_kill_C1 | 1'b0;

// read port to input operand mapping logic
assign ars_i_C1 = AR_rd0_data_C1;
assign art_i_C1 = AR_rd1_data_C1;
assign gs_i_C1 = gf_rd0_data_C1;
assign gt_i_C1 = gf_rd1_data_C1;
assign gr_i_C1 = gf_rd2_data_C1;

// logic to support verification
wire ignore_TIE_aWriteData_E = ~(AR_wd_def1_C0 & (TIE_arWrite_R | TIE_asWrite_R
| TIE_atWrite_R) & ~TIE_aDataKill_E);
wire ignore_TIE_aWriteData_M = ~(1'b0 & (TIE_arWrite_R | TIE_asWrite_R |
TIE_atWrite_R) & ~TIE_aDataKill_M);
wire ignore_TIE_bWriteData_E = (~TIE_btWrite_R & ~TIE_btWrite_R) |
TIE_bDataKill_E;
wire ignore_TIE_bWriteData16_E = ignore_TIE_bWriteData_E;
wire ignore_TIE_bWriteData8_E = ignore_TIE_bWriteData_E;
wire ignore_TIE_bWriteData4_E = ignore_TIE_bWriteData_E;
wire ignore_TIE_bWriteData2_E = ignore_TIE_bWriteData_E;
wire ignore_TIE_bWriteData1_E = ignore_TIE_bWriteData_E;
wire ignore_TIE_LSSize_R = ~TIE_Load_R & ~TIE_Store_R;
wire ignore_TIE_LSIndexed_R = ~TIE_Load_R & ~TIE_Store_R;
wire ignore_TIE_LSOffset_R = ~TIE_Load_R & ~TIE_Store_R | TIE_LSIndexed_R;
wire ignore_TIE_MemStoreData128_E = (TIE_LSSize_R != 5'b10000) | ~TIE_Store_R;
wire ignore_TIE_MemStoreData64_E = (TIE_LSSize_R != 5'b01000) | ~TIE_Store_R;
wire ignore_TIE_MemStoreData32_E = (TIE_LSSize_R != 5'b00100) | ~TIE_Store_R;
wire ignore_TIE_MemStoreData16_E = (TIE_LSSize_R != 5'b00010) | ~TIE_Store_R;
wire ignore_TIE_MemStoreData8_E = (TIE_LSSize_R != 5'b00001) | ~TIE_Store_R;

// clock and instructions
assign clk = G1WCLK;
assign Inst_C0 = Instr_R;
assign TIE_inst_R = TIE_Inst_C0;

// AR-related signals to/from core
assign TIE_asRead_R = ars_use_C0;
assign TIE_atRead_R = art_use_C0;
assign TIE_atWrite_R = art_def_C0;
assign TIE_arWrite_R = arr_def_C0;
assign TIE_asWrite_R = ars_def_C0;
assign TIE_aWriteM_R = 0;
assign TIE_aWriteData_E = ignore_TIE_aWriteData_E ? 0 : AR_wd_data32_C1;
assign TIE_aWriteData_M = ignore_TIE_aWriteData_M ? 0 : 0;
assign TIE_aDataKill_E = AR_wd_kill_C1;
assign TIE_aDataKill_M = 0;
assign AR_rd0_data_C1 = SBus_E;
assign AR_rd1_data_C1 = TBus_E;

// BR-related signals to/from core
assign TIE_bsRead_R = 1'b0 | bs_use_C0 | bs4_use_C0 | bs8_use_C0;
assign TIE_btRead_R = 1'b0 | bt_use_C0;
assign TIE_btWrite_R = 1'b0 | bt_def_C0;
assign TIE_bsWrite_R = 1'b0 | bs_def_C0 | bs4_def_C0 | bs8_def_C0;
assign TIE_brWrite_R = 1'b0 | br_def_C0;
```

```
assign TIE_bWriteData16_E = ignore_TIE_bWriteData16_E ? 0 : 0;
assign TIE_bWriteData8_E = ignore_TIE_bWriteData8_E ? 0 : 0;
assign TIE_bWriteData4_E = ignore_TIE_bWriteData4_E ? 0 : 0;
assign TIE_bWriteData2_E = ignore_TIE_bWriteData2_E ? 0 : 0;
assign TIE_bWriteData1_E = ignore_TIE_bWriteData1_E ? 0 : 0;
assign TIE_bDataKill_E = 0;
assign TIE_bWriteSize_R = {1'b0, 1'b0, 1'b0, 1'b0, 1'b0};
assign TIE_bsReadSize_R = {1'b0, 1'b0, 1'b0, 1'b0, 1'b0};
assign TIE_btReadSize_R = {1'b0, 1'b0, 1'b0, 1'b0, 1'b0};

// Load/store signals to/from core
assign TIE_Load_R = load_instruction_C0;
assign TIE_Store_R = store_instruction_C0;
assign TIE_LSSize_R = ignore_TIE_LSSize_R ? 0 : LSSize_C0;
assign TIE_LSIndexed_R = ignore_TIE_LSIndexed_R ? 0 : LSIndexed_C0;
assign TIE_LSOffset_R = ignore_TIE_LSOffset_R ? 0 : VAddrOffset_C0;
assign TIE_MemStoreData128_E = ignore_TIE_MemStoreData128_E ? 0 :
MemDataOut128_C1;
assign TIE_MemStoreData64_E = ignore_TIE_MemStoreData64_E ? 0 :
MemDataOut64_C1;
assign TIE_MemStoreData32_E = ignore_TIE_MemStoreData32_E ? 0 :
MemDataOut32_C1;
assign TIE_MemStoreData16_E = ignore_TIE_MemStoreData16_E ? 0 :
MemDataOut16_C1;
assign TIE_MemStoreData8_E = ignore_TIE_MemStoreData8_E ? 0 : MemDataOut8_C1;
assign MemDataIn128_C2 = TIE_MemLoadData_M;
assign MemDataIn64_C2 = TIE_MemLoadData_M;
assign MemDataIn32_C2 = TIE_MemLoadData_M;
assign MemDataIn16_C2 = TIE_MemLoadData_M;
assign MemDataIn8_C2 = TIE_MemLoadData_M;
assign VAddrIn_C1 = MemOpAddr_E;

// CPEnable and control signals to/from core
assign CPEnable_C1 = CPEnable;
assign TIE_Exception_E = Exception_C1;
assign TIE_ExcCause_E = ExcCause_C1;
assign KillPipe_W = Except_W | Replay_W;
endmodule
module xtdelay1(xtout, xtin, clk);
parameter size = 1;
output [size-1:0] xtout;
input [size-1:0] xtin;
input clk;
    assign xtout = xtin;
endmodule

module xtdelay2(xtout, xtin, clk);
parameter size = 1;
output [size-1:0] xtout;
input [size-1:0] xtin;
input clk;
    assign xtout = xtin;
endmodule

module xtRFenlatch(xtRFenlatchout,xtin,xten,clk);
   parameter size = 32;
   output [size-1:0] xtRFenlatchout;
```

```
   input [size-1:0]  xtin;
   input          xten;
   input          clk;

   reg [size-1:0]    xtRFenlatchout;

 always @(clk or xten or xtin or xtRFenlatchout) begin
  if (clk) begin
     xtRFenlatchout <= #1 (xten) ? xtin : xtRFenlatchout;
  end
 end

endmodule
module xtRFlatch(xtRFlatchout,xtin,clk);
   parameter size = 32;
   output [size-1:0] xtRFlatchout;
   input [size-1:0]  xtin;
   input          clk;

   reg [size-1:0]    xtRFlatchout;

 always @(clk or xtin) begin
  if (clk) begin
   xtRFlatchout <= #1 xtin;
  end
 end

endmodule
module xtadd(xtout, a, b);
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  a;
   input [size-1:0]  b;

   assign xtout = a + b;

endmodule
module xtaddc(sum, carry, a, b, c);
   parameter size = 32;

   output [size-1:0] sum;
   output          carry;
   input [size-1:0]  a;
   input [size-1:0]  b;
   input          c;

   wire           junk;

   assign {carry, sum, junk} = {a,c} + {b,c};

endmodule
module xtaddcin(xtout, a, b, c);
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  a;
```

```
   input [size-1:0]  b;
   input         c;

   assign xtout = ({a,c} + {b,c}) >> 1;

endmodule
module xtaddcout(sum, carry, a, b);
   parameter size = 1;

   output [size-1:0] sum;
   output            carry;
   input [size-1:0]  a;
   input [size-1:0]  b;

   assign {carry, sum} = a + b;

endmodule
module xtbooth(out, cin, a, b, sign, negate);
parameter size = 16;
output [size+1:0] out;
output cin;
input [size-1:0] a;
input [2:0] b;
input sign, negate;
    wire ase = sign & a[size-1];
    wire [size+1:0] ax1 = {ase, ase, a};
    wire [size+1:0] ax2 = {ase, a, 1'd0};
    wire one = b[1] ^ b[0];
    wire two = b[2] ? ~b[1] & ~b[0] : b[1] & b[0];
    wire cin = negate ? (~b[2] & (b[1] | b[0])) : (b[2] & ~(b[1] & b[0]));
    assign out = {size+2{cin}} ^ (ax1&{size+2{one}} | ax2&{size+2{two}});
endmodule
module xtclock_gate_nor(xtout,xtin1,xtin2);
 output xtout;
 input xtin1,xtin2;

 assign xtout = ~(xtin1 || xtin2);

endmodule
module xtclock_gate_or(xtout,xtin1,xtin2);
 output xtout;
 input xtin1,xtin2;

 assign xtout = (xtin1 || xtin2);

endmodule
module xtcsa (sum, carry, a, b, c);
   parameter size = 1;

   output [size-1:0]    sum;
   output [size-1:0]    carry;
   input [size-1:0]     a;
   input [size-1:0]     b;
   input [size-1:0]     c;

   assign sum = a ^ b ^ c;
   assign carry = (a & b) | (b & c) | (c & a) ;
```

```
endmodule
module xtenflop(xtout, xtin, en, clk);
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  xtin;
   input          en;
   input          clk;
   reg [size-1:0]    tmp;

   assign xtout = tmp;
   always @(posedge clk) begin
      if (en)
      tmp <= #1 xtin;
   end

endmodule
module xtfa(sum, carry, a, b, c);
output sum, carry;
input a, b, c;
    assign sum = a ^ b ^ c;
    assign carry = a & b | a & c | b & c;
endmodule
module xtflop(xtout, xtin, clk);
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  xtin;
   input          clk;
   reg [size-1:0]    tmp;

   assign xtout = tmp;
   always @(posedge clk) begin
      tmp <= #1 xtin;
   end

endmodule
module xtha(sum, carry, a, b);
output sum, carry;
input a, b;
    assign sum = a ^ b;
    assign carry = a & b;
endmodule
module xtinc(xtout, a);
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  a;

   assign xtout = a + 1;

endmodule
module xtmux2e(xtout, a, b, sel);
   parameter size = 32;

   output [size-1:0] xtout;
```

```
   input [size-1:0]  a;
   input [size-1:0]  b;
   input          sel;

   assign xtout = (~sel) ? a : b;

endmodule
module xtmux3e(xtout, a, b, c, sel);
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  a;
   input [size-1:0]  b;
   input [size-1:0]  c;
   input [1:0]          sel;
   reg [size-1:0]    xtout;


   always @(a or b or c or sel) begin
      xtout = sel[1] ? c : (sel[0] ? b : a);
   end
endmodule
module xtmux4e(xtout, a, b, c, d, sel);
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  a;
   input [size-1:0]  b;
   input [size-1:0]  c;
   input [size-1:0]  d;
   input [1:0]          sel;
   reg [size-1:0]    xtout;

   // synopsys infer_mux "xtmux4e"
   always @(sel or a or b or c or d) begin : xtmux4e
      case (sel)        // synopsys parallel_case full_case
      2'b00:
        xtout = a;
      2'b01:
        xtout = b;
      2'b10:
        xtout = c;
      2'b11:
        xtout = d;
      default:
        xtout = {size{1'bx}};
      endcase // case(sel)
   end // always @ (sel or a or b or c or d)

endmodule
module xtnflop(xtout, xtin, clk);
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  xtin;
   input          clk;
   reg [size-1:0]    tmp;
```

```
   assign xtout = tmp;
   always @(negedge clk) begin
      tmp <= #1 xtin;
   end // always @ (negedge clk)

endmodule
module xtscflop(xtout, xtin, clrb, clk);  // sync clear ff
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  xtin;
   input         clrb;
   input         clk;
   reg [size-1:0]    tmp;

   assign xtout = tmp;
   always @(posedge clk) begin
      if (!clrb) tmp <=  0;
      else tmp <= #1 xtin;
   end

endmodule
module xtscenflop(xtout, xtin, en, clrb, clk);  // sync clear
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  xtin;
   input         en;
   input             clrb;
   input         clk;
   reg [size-1:0]    tmp;

   assign xtout = tmp;
   always @(posedge clk) begin
      if (!clrb) tmp <= 0;
      else if (en)
      tmp <= #1 xtin;
   end

endmodule
```

verysys/verify_ref.v

```
module xmTIE_gf_Regfile(rd0_data_C1, rd0_addr_C0, rd0_width8_C0, rd0_use1_C0,
  rd1_data_C1, rd1_addr_C0, rd1_width8_C0, rd1_use1_C0, rd2_data_C1,
  rd2_addr_C0, rd2_width8_C0, rd2_use1_C0, wd_addr_C0, wd_width8_C0,
  wd_def1_C0, wd_def2_C0, wd_data8_C1, wd_data8_C2, wd_wen_C1, wd_wen_C2,
  Kill_E, KillPipe_W, Stall_R, clk);
    output [7:0] rd0_data_C1;
    input [3:0] rd0_addr_C0;
    input rd0_width8_C0;
    input rd0_use1_C0;
    output [7:0] rd1_data_C1;
    input [3:0] rd1_addr_C0;
    input rd1_width8_C0;
```

```
    input rd1_use1_C0;
    output [7:0] rd2_data_C1;
    input [3:0] rd2_addr_C0;
    input rd2_width8_C0;
    input rd2_use1_C0;
    input [3:0] wd_addr_C0;
    input wd_width8_C0;
    input wd_def1_C0;
    input wd_def2_C0;
    input [7:0] wd_data8_C1;
    input [7:0] wd_data8_C2;
    input wd_wen_C1;
    input wd_wen_C2;
    input Kill_E;
    input KillPipe_W;
    output Stall_R;
    input clk;


    /*************************************************************************
          READ PORT rd0
     *************************************************************************/
    // compute the address mask
    wire rd0_addr_mask_C0 = 1'd0;

    // masked address pipeline
    wire rd0_maddr_C0 = 1'd0;

    // bank-qualified use
    wire rd0_use1_bank0_C0 = (rd0_use1_C0 & (rd0_maddr_C0 == (1'd0 &
rd0_addr_mask_C0)));

    // alignment mux for use 1
    wire [7:0] rd0_data_bank0_C1;
    assign rd0_data_C1[7:0] = rd0_data_bank0_C1;


    /*************************************************************************
          READ PORT rd1
     *************************************************************************/
    // compute the address mask
    wire rd1_addr_mask_C0 = 1'd0;

    // masked address pipeline
    wire rd1_maddr_C0 = 1'd0;

    // bank-qualified use
    wire rd1_use1_bank0_C0 = (rd1_use1_C0 & (rd1_maddr_C0 == (1'd0 &
rd1_addr_mask_C0)));

    // alignment mux for use 1
    wire [7:0] rd1_data_bank0_C1;
    assign rd1_data_C1[7:0] = rd1_data_bank0_C1;


    /*************************************************************************
          READ PORT rd2
```

```
    ***********************************************************************/
    // compute the address mask
    wire rd2_addr_mask_C0 = 1'd0;

    // masked address pipeline
    wire rd2_maddr_C0 = 1'd0;

    // bank-qualified use
    wire rd2_use1_bank0_C0 = (rd2_use1_C0 & (rd2_maddr_C0 == (1'd0 &
rd2_addr_mask_C0)));

    // alignment mux for use 1
    wire [7:0] rd2_data_bank0_C1;
    assign rd2_data_C1[7:0] = rd2_data_bank0_C1;


    /***********************************************************************
          WRITE PORT wd
     ***********************************************************************/
    // compute the address mask
    wire wd_addr_mask_C0 = 1'd0;

    // bank-qualified write def for port wd
    wire wd_def1_bank0_C0 = (wd_def1_C0 & ((wd_addr_C0 & wd_addr_mask_C0) ==
(1'd0 & wd_addr_mask_C0)));
    wire wd_def2_bank0_C0 = (wd_def2_C0 & ((wd_addr_C0 & wd_addr_mask_C0) ==
(1'd0 & wd_addr_mask_C0)));

    // write mux for def 1
    wire [7:0] wd_wdata_C1;
    assign wd_wdata_C1 = {1{wd_data8_C1[7:0]}};

    // write mux for def 2
    wire [7:0] wd_wdata_C2;
    assign wd_wdata_C2 = {1{wd_data8_C2[7:0]}};


    wire Stall_R0;
    /***********************************************************************
          PIPELINED BANK
     ***********************************************************************/
    xmTIE_gf_Regfile_bank TIE_gf_Regfile_bank0(rd0_data_bank0_C1,
        rd0_addr_C0[3:0], rd0_use1_bank0_C0, rd1_data_bank0_C1,
rd1_addr_C0[3:0],
        rd1_use1_bank0_C0, rd2_data_bank0_C1, rd2_addr_C0[3:0],
rd2_use1_bank0_C0,
        wd_addr_C0[3:0], wd_def1_bank0_C0, wd_def2_bank0_C0, wd_wdata_C1[7:0],
        wd_wdata_C2[7:0], wd_wen_C1, wd_wen_C2, Kill_E, KillPipe_W, Stall_R0,
        clk);

    assign Stall_R = Stall_R0 | 1'b0;

endmodule


module xmTIE_gf_Regfile_bank(rd0_data_C1, rd0_addr_C0, rd0_use1_C0,
  rd1_data_C1, rd1_addr_C0, rd1_use1_C0, rd2_data_C1, rd2_addr_C0,
```

```
rd2_use1_C0, wd_addr_C0, wd_def1_C0, wd_def2_C0, wd_data_C1, wd_data_C2,
wd_wen_C1, wd_wen_C2, Kill_E, KillPipe_W, Stall_R, clk);
  output [7:0] rd0_data_C1;
  input [3:0] rd0_addr_C0;
  input rd0_use1_C0;
  output [7:0] rd1_data_C1;
  input [3:0] rd1_addr_C0;
  input rd1_use1_C0;
  output [7:0] rd2_data_C1;
  input [3:0] rd2_addr_C0;
  input rd2_use1_C0;
  input [3:0] wd_addr_C0;
  input wd_def1_C0;
  input wd_def2_C0;
  input [7:0] wd_data_C1;
  input [7:0] wd_data_C2;
  input wd_wen_C1;
  input wd_wen_C2;
  input Kill_E;
  input KillPipe_W;
  output Stall_R;
  input clk;

  wire rd0_use2_C0 = 1'd0;
  wire rd1_use2_C0 = 1'd0;
  wire rd2_use2_C0 = 1'd0;

  wire kill_C0 = KillPipe_W;
  wire kill_C1 = KillPipe_W | Kill_E;
  wire kill_C2 = KillPipe_W;
  wire kill_C3 = KillPipe_W;

  // write definition pipeline
  wire wd_ns_def1_C0 = wd_def1_C0 & 1'b1 & ~kill_C0;
  wire wd_def1_C1;
  xtdelay1 #(1) iwd_def1_C1(wd_def1_C1, wd_ns_def1_C0, clk);
  wire wd_ns_def2_C0 = wd_def2_C0 & 1'b1 & ~kill_C0;
  wire wd_def2_C1;
  xtdelay1 #(1) iwd_def2_C1(wd_def2_C1, wd_ns_def2_C0, clk);
  wire wd_ns_def2_C1 = wd_def2_C1 & wd_wen_C1 & ~kill_C1;
  wire wd_def2_C2;
  xtdelay1 #(1) iwd_def2_C2(wd_def2_C2, wd_ns_def2_C1, clk);

  // write enable pipeline
  wire wd_we_C2;
  wire wd_we_C3;
  wire wd_ns_we_C1 = (1'd0 | (wd_def1_C1 & wd_wen_C1)) & ~kill_C1;
  wire wd_ns_we_C2 = (wd_we_C2 | (wd_def2_C2 & wd_wen_C2)) & ~kill_C2;
  wire wd_ns_we_C3 = (wd_we_C3 | (1'd0 & 1'd0)) & ~kill_C3;
  xtdelay1 #(1) iwd_we_C2(wd_we_C2, wd_ns_we_C1, clk);
  xtdelay1 #(1) iwd_we_C3(wd_we_C3, wd_ns_we_C2, clk);

  // write address pipeline
  wire [3:0] wd_addr_C1;
  wire [3:0] wd_addr_C2;
  wire [3:0] wd_addr_C3;
  xtdelay1 #(4) iwd_addr_C1(wd_addr_C1, wd_addr_C0, clk);
```

```
    xtdelay1 #(4) iwd_addr_C2(wd_addr_C2, wd_addr_C1, clk);
    xtdelay1 #(4) iwd_addr_C3(wd_addr_C3, wd_addr_C2, clk);

    // write data pipeline
    wire [7:0] wd_result_C2;
    wire [7:0] wd_result_C3;
    wire [7:0] wd_mux_C1 = wd_data_C1;
    wire [7:0] wd_mux_C2 = wd_def2_C2 ? wd_data_C2 : wd_result_C2;
    xtdelay1 #(8) iwd_result_C2(wd_result_C2, wd_mux_C1, clk);
    xtdelay1 #(8) iwd_result_C3(wd_result_C3, wd_mux_C2, clk);

    wire [7:0] rd0_data_C0;
    wire [7:0] rd1_data_C0;
    wire [7:0] rd2_data_C0;

    xtdelay1 #(8) ird0_data_C1(rd0_data_C1, rd0_data_C0, clk);

    xtdelay1 #(8) ird1_data_C1(rd1_data_C1, rd1_data_C0, clk);

    xtdelay1 #(8) ird2_data_C1(rd2_data_C1, rd2_data_C0, clk);

    assign Stall_R =
        ((wd_addr_C1 == rd0_addr_C0) & (
            (rd0_use1_C0 & (wd_ns_def2_C1)))) |
        ((wd_addr_C1 == rd1_addr_C0) & (
            (rd1_use1_C0 & (wd_ns_def2_C1)))) |
        ((wd_addr_C1 == rd2_addr_C0) & (
            (rd2_use1_C0 & (wd_ns_def2_C1)))) |
        1'b0;

    // verification register file replacement
    wire [7:0] xwd_verify;
    xtenflop #(8) wd_verify(xwd_verify, wd_result_C3, wd_ns_we_C3, clk);
    xtflop #(8) rd0_verify(rd0_data_C0, xwd_verify, clk);
    xtflop #(8) rd1_verify(rd1_data_C0, xwd_verify, clk);
    xtflop #(8) rd2_verify(rd2_data_C0, xwd_verify, clk);
endmodule


module xmTIE_gfmod_State(ps_data_C1, ps_width8_C0, ps_use1_C0, ns_width8_C0,
  ns_def1_C0, ns_data8_C1, ns_wen_C1, Kill_E, KillPipe_W, Stall_R, clk);
    output [7:0] ps_data_C1;
    input ps_width8_C0;
    input ps_use1_C0;
    input ns_width8_C0;
    input ns_def1_C0;
    input [7:0] ns_data8_C1;
    input ns_wen_C1;
    input Kill_E;
    input KillPipe_W;
    output Stall_R;
    input clk;

    wire ps_addr_C0 = 1'd0;
    wire ns_addr_C0 = 1'd0;
    wire ns_wen_C2 = 1'd1;
```

```
/************************************************************************
      READ PORT ps
 ************************************************************************/
// compute the address mask
wire ps_addr_mask_C0 = 1'd0;

// masked address pipeline
wire ps_maddr_C0 = 1'd0;

// bank-qualified use
wire ps_use1_bank0_C0 = (ps_use1_C0 & (ps_maddr_C0 == (1'd0 &
ps_addr_mask_C0)));

// alignment mux for use 1
wire [7:0] ps_data_bank0_C1;
assign ps_data_C1[7:0] = ps_data_bank0_C1;


/************************************************************************
      WRITE PORT ns
 ************************************************************************/
// compute the address mask
wire ns_addr_mask_C0 = 1'd0;

// bank-qualified write def for port ns
wire ns_def1_bank0_C0 = (ns_def1_C0 & ((ns_addr_C0 & ns_addr_mask_C0) ==
(1'd0 & ns_addr_mask_C0)));

// write mux for def 1
wire [7:0] ns_wdata_C1;
assign ns_wdata_C1 = {1{ns_data8_C1[7:0]}};


wire Stall_R0;
/************************************************************************
      PIPELINED BANK
 ************************************************************************/
xmTIE_gfmod_State_bank TIE_gfmod_State_bank0(ps_data_bank0_C1,
    ps_use1_bank0_C0, ns_def1_bank0_C0, ns_wdata_C1[7:0], ns_wen_C1,
    ns_wen_C2, Kill_E, KillPipe_W, Stall_R0, clk);

assign Stall_R = Stall_R0 | 1'b0;

endmodule


module xmTIE_gfmod_State_bank(ps_data_C1, ps_use1_C0, ns_def1_C0, ns_data_C1,
  ns_wen_C1, ns_wen_C2, Kill_E, KillPipe_W, Stall_R, clk);
    output [7:0] ps_data_C1;
    input ps_use1_C0;
    input ns_def1_C0;
    input [7:0] ns_data_C1;
    input ns_wen_C1;
    input ns_wen_C2;
    input Kill_E;
    input KillPipe_W;
    output Stall_R;
```

```
input clk;

wire ps_addr_C0 = 1'd0;
wire ps_use2_C0 = 1'd0;
wire ns_addr_C0 = 1'd0;
wire ns_def2_C0 = 1'd0;
wire [7:0] ns_data_C2 = 0;

wire kill_C0 = KillPipe_W;
wire kill_C1 = KillPipe_W | Kill_E;
wire kill_C2 = KillPipe_W;
wire kill_C3 = KillPipe_W;

// write definition pipeline
wire ns_ns_def1_C0 = ns_def1_C0 & 1'b1 & ~kill_C0;
wire ns_def1_C1;
xtdelay1 #(1) ins_def1_C1(ns_def1_C1, ns_ns_def1_C0, clk);
wire ns_ns_def2_C0 = 1'd0;
wire ns_def2_C1 = 1'd0;
wire ns_ns_def2_C1 = 1'd0;
wire ns_def2_C2 = 1'd0;

// write enable pipeline
wire ns_we_C2;
wire ns_we_C3;
wire ns_ns_we_C1 = (1'd0 | (ns_def1_C1 & ns_wen_C1)) & ~kill_C1;
wire ns_ns_we_C2 = (ns_we_C2 | (ns_def2_C2 & ns_wen_C2)) & ~kill_C2;
wire ns_ns_we_C3 = (ns_we_C3 | (1'd0 & 1'd0)) & ~kill_C3;
xtdelay1 #(1) ins_we_C2(ns_we_C2, ns_ns_we_C1, clk);
xtdelay1 #(1) ins_we_C3(ns_we_C3, ns_ns_we_C2, clk);

// write address pipeline
wire  ns_addr_C1;
wire  ns_addr_C2;
wire  ns_addr_C3;
assign ns_addr_C1 = 1'd0;
assign ns_addr_C2 = 1'd0;
assign ns_addr_C3 = 1'd0;

// write data pipeline
wire [7:0] ns_result_C2;
wire [7:0] ns_result_C3;
wire [7:0] ns_mux_C1 = ns_data_C1;
wire [7:0] ns_mux_C2 = ns_def2_C2 ? ns_data_C2 : ns_result_C2;
xtdelay1 #(8) ins_result_C2(ns_result_C2, ns_mux_C1, clk);
xtdelay1 #(8) ins_result_C3(ns_result_C3, ns_mux_C2, clk);

wire [7:0] ps_data_C0;

xtdelay1 #(8) ips_data_C1(ps_data_C1, ps_data_C0, clk);

assign Stall_R =
    ((ns_addr_C1 == ps_addr_C0) & (
        (ps_use1_C0 & (ns_ns_def2_C1)))) |
    1'b0;

// verification register file replacement
```

```
    wire [7:0] xns_verify;
    xtenflop #(8) ns_verify(xns_verify, ns_result_C3, ns_ns_we_C3, clk);
    xtflop #(8) ps_verify(ps_data_C0, xns_verify, clk);
endmodule


module xmTIE_decoder (
GFADD8,
GFADD8I,
GFMULX8,
GFRWMOD8,
LGF8_I,
SGF8_I,
LGF8_IU,
SGF8_IU,
LGF8_X,
SGF8_X,
LGF8_XU,
SGF8_XU,
RUR0,
WUR0,
imm4,
imm8,
art_use,
art_def,
ars_use,
ars_def,
arr_use,
arr_def,
br_use,
br_def,
bs_use,
bs_def,
bt_use,
bt_def,
bs4_use,
bs4_def,
bs8_use,
bs8_def,
gr_use,
gr_def,
gs_use,
gs_def,
gt_use,
gt_def,
gfmod_use1,
gfmod_def1,
AR_rd0_use1,
AR_rd0_width32,
AR_rd1_use1,
AR_rd1_width32,
AR_wd_def1,
AR_wd_width32,
gf_rd0_addr,
gf_rd0_use1,
gf_rd0_width8,
gf_rd1_addr,
```

```
gf_rd1_use1,
gf_rd1_width8,
gf_rd2_addr,
gf_rd2_use1,
gf_rd2_width8,
gf_wd_addr,
gf_wd_def2,
gf_wd_def1,
gf_wd_width8,
GFADD8_semantic,
GFADD8I_semantic,
GFMULX8_semantic,
GFRWMOD8_semantic,
LGF8_I_semantic,
LGF8_IU_semantic,
LGF8_X_semantic,
LGF8_XU_semantic,
SGF8_I_semantic,
SGF8_IU_semantic,
SGF8_X_semantic,
SGF8_XU_semantic,
RUR0_semantic,
WUR0_semantic,
load_instruction,
store_instruction,
TIE_Inst,
Inst
);
output GFADD8;
output GFADD8I;
output GFMULX8;
output GFRWMOD8;
output LGF8_I;
output SGF8_I;
output LGF8_IU;
output SGF8_IU;
output LGF8_X;
output SGF8_X;
output LGF8_XU;
output SGF8_XU;
output RUR0;
output WUR0;
output [31:0] imm4;
output [7:0] imm8;
output art_use;
output art_def;
output ars_use;
output ars_def;
output arr_use;
output arr_def;
output br_use;
output br_def;
output bs_use;
output bs_def;
output bt_use;
output bt_def;
output bs4_use;
```

```
output bs4_def;
output bs8_use;
output bs8_def;
output gr_use;
output gr_def;
output gs_use;
output gs_def;
output gt_use;
output gt_def;
output gfmod_use1;
output gfmod_def1;
output AR_rd0_use1;
output AR_rd0_width32;
output AR_rd1_use1;
output AR_rd1_width32;
output AR_wd_def1;
output AR_wd_width32;
output [3:0] gf_rd0_addr;
output gf_rd0_use1;
output gf_rd0_width8;
output [3:0] gf_rd1_addr;
output gf_rd1_use1;
output gf_rd1_width8;
output [3:0] gf_rd2_addr;
output gf_rd2_use1;
output gf_rd2_width8;
output [3:0] gf_wd_addr;
output gf_wd_def2;
output gf_wd_def1;
output gf_wd_width8;
output GFADD8_semantic;
output GFADD8I_semantic;
output GFMULX8_semantic;
output GFRWMOD8_semantic;
output LGF8_I_semantic;
output LGF8_IU_semantic;
output LGF8_X_semantic;
output LGF8_XU_semantic;
output SGF8_I_semantic;
output SGF8_IU_semantic;
output SGF8_X_semantic;
output SGF8_XU_semantic;
output RUR0_semantic;
output WUR0_semantic;
output load_instruction;
output store_instruction;
output TIE_Inst;
input [23:0] Inst;

wire [3:0] op2 = {Inst[23:20]};
wire [3:0] op1 = {Inst[19:16]};
wire [3:0] op0 = {Inst[3:0]};
wire QRST = (op0==4'b0000);
wire CUST0 = (op1==4'b0110) & QRST;
assign GFADD8 = (op2==4'b0000) & CUST0;
assign GFADD8I = (op2==4'b0100) & CUST0;
assign GFMULX8 = (op2==4'b0001) & CUST0;
```

```
assign GFRWMOD8 = (op2==4'b0010) & CUST0;
wire [3:0] r = {Inst[15:12]};
wire LSCI = (op0==4'b0011);
assign LGF8_I = (r==4'b0000) & LSCI;
assign SGF8_I = (r==4'b0001) & LSCI;
assign LGF8_IU = (r==4'b0010) & LSCI;
assign SGF8_IU = (r==4'b0011) & LSCI;
wire LSCX = (op1==4'b1000) & QRST;
assign LGF8_X = (op2==4'b0000) & LSCX;
assign SGF8_X = (op2==4'b0001) & LSCX;
assign LGF8_XU = (op2==4'b0010) & LSCX;
assign SGF8_XU = (op2==4'b0011) & LSCX;
wire [3:0] s = {Inst[11:8]};
wire [3:0] t = {Inst[7:4]};
wire [7:0] st = {s,t};
wire RST3 = (op1==4'b0011) & QRST;
wire RUR = (op2==4'b1110) & RST3;
assign RUR0 = (st==8'b00000000) & RUR;
wire [7:0] sr = {r,s};
wire WUR = (op2==4'b1111) & RST3;
assign WUR0 = (sr==8'b00000000) & WUR;
assign gfmod_use1 = GFMULX8 | GFRWMOD8 | RUR0 | 1'b0;
assign gfmod_def1 = GFRWMOD8 | WUR0 | 1'b0;
assign AR_rd0_use1 = 1'b0
    | LGF8_I
    | SGF8_I
    | LGF8_IU
    | SGF8_IU
    | LGF8_X
    | SGF8_X
    | LGF8_XU
    | SGF8_XU;
assign AR_rd0_width32 = 1'b0;
assign AR_rd1_use1 = 1'b0
    | LGF8_X
    | SGF8_X
    | LGF8_XU
    | SGF8_XU
    | WUR0;
assign AR_rd1_width32 = 1'b0;
assign AR_wd_def1 = 1'b0
    | LGF8_IU
    | SGF8_IU
    | LGF8_XU
    | SGF8_XU
    | RUR0;
assign AR_wd_width32 = 1'b0;
assign gf_rd0_use1 = 1'b0
    | GFADD8
    | GFADD8I
    | GFMULX8;
assign gf_rd0_width8 = 1'b0;
assign gf_rd1_use1 = 1'b0
    | GFADD8
    | GFRWMOD8
    | SGF8_I
    | SGF8_IU;
```

```
assign gf_rd1_width8 = 1'b0;
assign gf_rd2_use1 = 1'b0
     | SGF8_X
     | SGF8_XU;
assign gf_rd2_width8 = 1'b0;
assign gf_wd_def2 = 1'b0
     | LGF8_I
     | LGF8_IU
     | LGF8_X
     | LGF8_XU;
assign gf_wd_def1 = 1'b0
     | GFADD8
     | GFADD8I
     | GFMULX8
     | GFRWMOD8;
assign gf_wd_width8 = 1'b0;
assign art_def = 1'b0;
assign art_use = LGF8_X | SGF8_X | LGF8_XU | SGF8_XU | WUR0 | 1'b0;
assign ars_def = LGF8_IU | SGF8_IU | LGF8_XU | SGF8_XU | 1'b0;
assign ars_use = LGF8_I | SGF8_I | LGF8_IU | SGF8_IU | LGF8_X | SGF8_X |
LGF8_XU | SGF8_XU | 1'b0;
assign arr_def = RUR0 | 1'b0;
assign arr_use = 1'b0;
assign br_def = 1'b0;
assign br_use = 1'b0;
assign bs_def = 1'b0;
assign bs_use = 1'b0;
assign bt_def = 1'b0;
assign bt_use = 1'b0;
assign bs4_def = 1'b0;
assign bs4_use = 1'b0;
assign bs8_def = 1'b0;
assign bs8_use = 1'b0;
assign gr_def = GFADD8 | GFADD8I | GFMULX8 | LGF8_X | LGF8_XU | 1'b0;
assign gr_use = SGF8_X | SGF8_XU | 1'b0;
assign gs_def = 1'b0;
assign gs_use = GFADD8 | GFADD8I | GFMULX8 | 1'b0;
assign gt_def = GFRWMOD8 | LGF8_I | LGF8_IU | 1'b0;
assign gt_use = GFADD8 | GFRWMOD8 | SGF8_I | SGF8_IU | 1'b0;
wire [3:0] gr_addr = r;
wire [3:0] gs_addr = s;
wire [3:0] gt_addr = t;
assign gf_wd_addr = 4'b0
     | {4{gr_def}} & gr_addr
     | {4{gt_def}} & gt_addr;
assign gf_rd0_addr = gs_addr;
assign gf_rd1_addr = gt_addr;
assign gf_rd2_addr = gr_addr;
assign GFADD8_semantic = GFADD8 | 1'b0;
assign GFADD8I_semantic = GFADD8I | 1'b0;
assign GFMULX8_semantic = GFMULX8 | 1'b0;
assign GFRWMOD8_semantic = GFRWMOD8 | 1'b0;
assign LGF8_I_semantic = LGF8_I | 1'b0;
assign LGF8_IU_semantic = LGF8_IU | 1'b0;
assign LGF8_X_semantic = LGF8_X | 1'b0;
assign LGF8_XU_semantic = LGF8_XU | 1'b0;
assign SGF8_I_semantic = SGF8_I | 1'b0;
```

```
assign SGF8_IU_semantic = SGF8_IU | 1'b0;
assign SGF8_X_semantic = SGF8_X | 1'b0;
assign SGF8_XU_semantic = SGF8_XU | 1'b0;
assign RURO_semantic = RURO | 1'b0;
assign WURO_semantic = WURO | 1'b0;
assign imm4 = t;
wire [7:0] imm8 = {Inst[23:16]};
assign load_instruction = 1'b0
     | LGF8_I
     | LGF8_IU
     | LGF8_X
     | LGF8_XU;
assign store_instruction = 1'b0
     | SGF8_I
     | SGF8_IU
     | SGF8_X
     | SGF8_XU;
assign TIE_Inst = 1'b0
     | GFADD8
     | GFADD8I
     | GFMULX8
     | GFRWMOD8
     | LGF8_I
     | SGF8_I
     | LGF8_IU
     | SGF8_IU
     | LGF8_X
     | SGF8_X
     | LGF8_XU
     | SGF8_XU
     | RURO
     | WURO;
endmodule

module xmTIE_GFADD8 (
GFADD8_C0,
gr_o_C1,
gr_kill_C1,
gs_i_C1,
gt_i_C1,
clk
);
input GFADD8_C0;
output [7:0] gr_o_C1;
output gr_kill_C1;
input [7:0] gs_i_C1;
input [7:0] gt_i_C1;
input clk;
assign gr_o_C1 = (gs_i_C1) ^ (gt_i_C1);
wire GFADD8_C1;
xtdelay1 #(1) iGFADD8_C1(.xtin(GFADD8_C0), .xtout(GFADD8_C1), .clk(clk));
assign gr_kill_C1 = (1'b0) & (GFADD8_C1);
endmodule

module xmTIE_GFADD8I (
GFADD8I_C0,
gr_o_C1,
```

```
gr_kill_C1,
gs_i_C1,
imm4_C0,
clk
);
input GFADD8I_C0;
output [7:0] gr_o_C1;
output gr_kill_C1;
input [7:0] gs_i_C1;
input [31:0] imm4_C0;
input clk;
wire [31:0] imm4_C1;
xtdelay1 #(32) iimm4_C1(.xtin(imm4_C0), .xtout(imm4_C1), .clk(clk));
assign gr_o_C1 = (gs_i_C1) ^ (imm4_C1);
wire GFADD8I_C1;
xtdelay1 #(1) iGFADD8I_C1(.xtin(GFADD8I_C0), .xtout(GFADD8I_C1), .clk(clk));
assign gr_kill_C1 = (1'b0) & (GFADD8I_C1);
endmodule

module xmTIE_GFMULX8 (
GFMULX8_C0,
gr_o_C1,
gr_kill_C1,
gs_i_C1,
gfmod_ps_C1,
clk
);
input GFMULX8_C0;
output [7:0] gr_o_C1;
output gr_kill_C1;
input [7:0] gs_i_C1;
input [7:0] gfmod_ps_C1;
input clk;
assign gr_o_C1 = (gs_i_C1[7]) ? (({gs_i_C1[6:0], 1'b0}) ^ (gfmod_ps_C1)) :
({gs_i_C1[6:0], 1'b0});
wire GFMULX8_C1;
xtdelay1 #(1) iGFMULX8_C1(.xtin(GFMULX8_C0), .xtout(GFMULX8_C1), .clk(clk));
assign gr_kill_C1 = (1'b0) & (GFMULX8_C1);
endmodule

module xmTIE_GFRWMOD8 (
GFRWMOD8_C0,
gt_i_C1,
gt_o_C1,
gt_kill_C1,
gfmod_ps_C1,
gfmod_ns_C1,
gfmod_kill_C1,
clk
);
input GFRWMOD8_C0;
input [7:0] gt_i_C1;
output [7:0] gt_o_C1;
output gt_kill_C1;
input [7:0] gfmod_ps_C1;
output [7:0] gfmod_ns_C1;
output gfmod_kill_C1;
```

```
input clk;
wire [7:0] t1_C1;
assign t1_C1 = gt_i_C1;
wire [7:0] t2_C1;
assign t2_C1 = gfmod_ps_C1;
assign gfmod_ns_C1 = t1_C1;
assign gt_o_C1 = t2_C1;
wire GFRWMOD8_C1;
xtdelay1 #(1) iGFRWMOD8_C1(.xtin(GFRWMOD8_C0), .xtout(GFRWMOD8_C1), .clk(clk));
assign gfmod_kill_C1 = (1'b0) & (GFRWMOD8_C1);
assign gt_kill_C1 = (1'b0) & (GFRWMOD8_C1);
endmodule

module xmTIE_LGF8_I (
LGF8_I_C0,
gt_o_C2,
gt_kill_C2,
ars_i_C1,
imm8_C0,
MemDataIn8_C2,
LSSize_C0,
VAddrBase_C1,
VAddrOffset_C0,
LSIndexed_C0,
clk
);
input LGF8_I_C0;
output [7:0] gt_o_C2;
output gt_kill_C2;
input [31:0] ars_i_C1;
input [7:0] imm8_C0;
input [7:0] MemDataIn8_C2;
output [4:0] LSSize_C0;
output [31:0] VAddrBase_C1;
output [31:0] VAddrOffset_C0;
output LSIndexed_C0;
input clk;
assign LSSize_C0 = 32'h1;
assign VAddrBase_C1 = ars_i_C1;
assign LSIndexed_C0 = 1'b0;
assign VAddrOffset_C0 = imm8_C0;
assign gt_o_C2 = MemDataIn8_C2;
wire LGF8_I_C2;
xtdelay2 #(1) iLGF8_I_C2(.xtin(LGF8_I_C0), .xtout(LGF8_I_C2), .clk(clk));
assign gt_kill_C2 = (1'b0) & (LGF8_I_C2);
endmodule

module xmTIE_LGF8_IU (
LGF8_IU_C0,
gt_o_C2,
gt_kill_C2,
ars_i_C1,
ars_o_C1,
ars_kill_C1,
imm8_C0,
MemDataIn8_C2,
VAddrIn_C1,
```

```
LSSize_C0,
VAddrBase_C1,
VAddrOffset_C0,
LSIndexed_C0,
clk
);
input LGF8_IU_C0;
output [7:0] gt_o_C2;
output gt_kill_C2;
input [31:0] ars_i_C1;
output [31:0] ars_o_C1;
output ars_kill_C1;
input [7:0] imm8_C0;
input [7:0] MemDataIn8_C2;
input [31:0] VAddrIn_C1;
output [4:0] LSSize_C0;
output [31:0] VAddrBase_C1;
output [31:0] VAddrOffset_C0;
output LSIndexed_C0;
input clk;
assign LSSize_C0 = 32'h1;
assign VAddrBase_C1 = ars_i_C1;
assign LSIndexed_C0 = 1'b0;
assign VAddrOffset_C0 = imm8_C0;
assign gt_o_C2 = MemDataIn8_C2;
assign ars_o_C1 = VAddrIn_C1;
wire LGF8_IU_C2;
xtdelay2 #(1) iLGF8_IU_C2(.xtin(LGF8_IU_C0), .xtout(LGF8_IU_C2), .clk(clk));
assign gt_kill_C2 = (1'b0) & (LGF8_IU_C2);
wire LGF8_IU_C1;
xtdelay1 #(1) iLGF8_IU_C1(.xtin(LGF8_IU_C0), .xtout(LGF8_IU_C1), .clk(clk));
assign ars_kill_C1 = (1'b0) & (LGF8_IU_C1);
endmodule

module xmTIE_LGF8_X (
LGF8_X_C0,
gr_o_C2,
gr_kill_C2,
ars_i_C1,
art_i_C1,
MemDataIn8_C2,
VAddrIn_C1,
LSSize_C0,
VAddrBase_C1,
VAddrIndex_C1,
LSIndexed_C0,
clk
);
input LGF8_X_C0;
output [7:0] gr_o_C2;
output gr_kill_C2;
input [31:0] ars_i_C1;
input [31:0] art_i_C1;
input [7:0] MemDataIn8_C2;
input [31:0] VAddrIn_C1;
output [4:0] LSSize_C0;
output [31:0] VAddrBase_C1;
```

```
output [31:0] VAddrIndex_C1;
output LSIndexed_C0;
input clk;
assign LSSize_C0 = 32'h1;
assign VAddrBase_C1 = ars_i_C1;
assign LSIndexed_C0 = 1'b1;
assign VAddrIndex_C1 = art_i_C1;
assign gr_o_C2 = MemDataIn8_C2;
assign ars_o_C1 = VAddrIn_C1;
wire LGF8_X_C2;
xtdelay2 #(1) iLGF8_X_C2(.xtin(LGF8_X_C0), .xtout(LGF8_X_C2), .clk(clk));
assign gr_kill_C2 = (1'b0) & (LGF8_X_C2);
endmodule

module xmTIE_LGF8_XU (
LGF8_XU_C0,
gr_o_C2,
gr_kill_C2,
ars_i_C1,
ars_o_C1,
ars_kill_C1,
art_i_C1,
MemDataIn8_C2,
VAddrIn_C1,
LSSize_C0,
VAddrBase_C1,
VAddrIndex_C1,
LSIndexed_C0,
clk
);
input LGF8_XU_C0;
output [7:0] gr_o_C2;
output gr_kill_C2;
input [31:0] ars_i_C1;
output [31:0] ars_o_C1;
output ars_kill_C1;
input [31:0] art_i_C1;
input [7:0] MemDataIn8_C2;
input [31:0] VAddrIn_C1;
output [4:0] LSSize_C0;
output [31:0] VAddrBase_C1;
output [31:0] VAddrIndex_C1;
output LSIndexed_C0;
input clk;
assign LSSize_C0 = 32'h1;
assign VAddrBase_C1 = ars_i_C1;
assign LSIndexed_C0 = 1'b1;
assign VAddrIndex_C1 = art_i_C1;
assign gr_o_C2 = MemDataIn8_C2;
assign ars_o_C1 = VAddrIn_C1;
wire LGF8_XU_C2;
xtdelay2 #(1) iLGF8_XU_C2(.xtin(LGF8_XU_C0), .xtout(LGF8_XU_C2), .clk(clk));
assign gr_kill_C2 = (1'b0) & (LGF8_XU_C2);
wire LGF8_XU_C1;
xtdelay1 #(1) iLGF8_XU_C1(.xtin(LGF8_XU_C0), .xtout(LGF8_XU_C1), .clk(clk));
assign ars_kill_C1 = (1'b0) & (LGF8_XU_C1);
endmodule
```

```
module xmTIE_SGF8_I (
SGF8_I_C0,
gt_i_C1,
ars_i_C1,
imm8_C0,
LSSize_C0,
MemDataOut8_C1,
VAddrBase_C1,
VAddrOffset_C0,
LSIndexed_C0,
clk
);
input SGF8_I_C0;
input [7:0] gt_i_C1;
input [31:0] ars_i_C1;
input [7:0] imm8_C0;
output [4:0] LSSize_C0;
output [7:0] MemDataOut8_C1;
output [31:0] VAddrBase_C1;
output [31:0] VAddrOffset_C0;
output LSIndexed_C0;
input clk;
assign LSSize_C0 = 32'h1;
assign VAddrBase_C1 = ars_i_C1;
assign LSIndexed_C0 = 1'b0;
assign VAddrOffset_C0 = imm8_C0;
assign MemDataOut8_C1 = gt_i_C1;
endmodule

module xmTIE_SGF8_IU (
SGF8_IU_C0,
gt_i_C1,
ars_i_C1,
ars_o_C1,
ars_kill_C1,
imm8_C0,
VAddrIn_C1,
LSSize_C0,
MemDataOut8_C1,
VAddrBase_C1,
VAddrOffset_C0,
LSIndexed_C0,
clk
);
input SGF8_IU_C0;
input [7:0] gt_i_C1;
input [31:0] ars_i_C1;
output [31:0] ars_o_C1;
output ars_kill_C1;
input [7:0] imm8_C0;
input [31:0] VAddrIn_C1;
output [4:0] LSSize_C0;
output [7:0] MemDataOut8_C1;
output [31:0] VAddrBase_C1;
output [31:0] VAddrOffset_C0;
output LSIndexed_C0;
```

```
input clk;
assign LSSize_C0 = 32'h1;
assign VAddrBase_C1 = ars_i_C1;
assign LSIndexed_C0 = 1'b0;
assign VAddrOffset_C0 = imm8_C0;
assign MemDataOut8_C1 = gt_i_C1;
assign ars_o_C1 = VAddrIn_C1;
wire SGF8_IU_C1;
xtdelay1 #(1) iSGF8_IU_C1(.xtin(SGF8_IU_C0), .xtout(SGF8_IU_C1), .clk(clk));
assign ars_kill_C1 = (1'b0) & (SGF8_IU_C1);
endmodule

module xmTIE_SGF8_X (
SGF8_X_C0,
gr_i_C1,
ars_i_C1,
art_i_C1,
LSSize_C0,
MemDataOut8_C1,
VAddrBase_C1,
VAddrIndex_C1,
LSIndexed_C0,
clk
);
input SGF8_X_C0;
input [7:0] gr_i_C1;
input [31:0] ars_i_C1;
input [31:0] art_i_C1;
output [4:0] LSSize_C0;
output [7:0] MemDataOut8_C1;
output [31:0] VAddrBase_C1;
output [31:0] VAddrIndex_C1;
output LSIndexed_C0;
input clk;
assign LSSize_C0 = 32'h1;
assign VAddrBase_C1 = ars_i_C1;
assign LSIndexed_C0 = 1'b1;
assign VAddrIndex_C1 = art_i_C1;
assign MemDataOut8_C1 = gr_i_C1;
endmodule

module xmTIE_SGF8_XU (
SGF8_XU_C0,
gr_i_C1,
ars_i_C1,
ars_o_C1,
ars_kill_C1,
art_i_C1,
VAddrIn_C1,
LSSize_C0,
MemDataOut8_C1,
VAddrBase_C1,
VAddrIndex_C1,
LSIndexed_C0,
clk
);
input SGF8_XU_C0;
```

```
input [7:0] gr_i_C1;
input [31:0] ars_i_C1;
output [31:0] ars_o_C1;
output ars_kill_C1;
input [31:0] art_i_C1;
input [31:0] VAddrIn_C1;
output [4:0] LSSize_C0;
output [7:0] MemDataOut8_C1;
output [31:0] VAddrBase_C1;
output [31:0] VAddrIndex_C1;
output LSIndexed_C0;
input clk;
assign LSSize_C0 = 32'h1;
assign VAddrBase_C1 = ars_i_C1;
assign LSIndexed_C0 = 1'b1;
assign VAddrIndex_C1 = art_i_C1;
assign MemDataOut8_C1 = gr_i_C1;
assign ars_o_C1 = VAddrIn_C1;
wire SGF8_XU_C1;
xtdelay1 #(1) iSGF8_XU_C1(.xtin(SGF8_XU_C0), .xtout(SGF8_XU_C1), .clk(clk));
assign ars_kill_C1 = (1'b0) & (SGF8_XU_C1);
endmodule

module xmTIE_RUR0 (
RUR0_C0,
arr_o_C1,
arr_kill_C1,
gfmod_ps_C1,
clk
);
input RUR0_C0;
output [31:0] arr_o_C1;
output arr_kill_C1;
input [7:0] gfmod_ps_C1;
input clk;
assign arr_o_C1 = {gfmod_ps_C1};
wire RUR0_C1;
xtdelay1 #(1) iRUR0_C1(.xtin(RUR0_C0), .xtout(RUR0_C1), .clk(clk));
assign arr_kill_C1 = (1'b0) & (RUR0_C1);
endmodule

module xmTIE_WUR0 (
WUR0_C0,
art_i_C1,
gfmod_ns_C1,
gfmod_kill_C1,
clk
);
input WUR0_C0;
input [31:0] art_i_C1;
output [7:0] gfmod_ns_C1;
output gfmod_kill_C1;
input clk;
assign gfmod_ns_C1 = {art_i_C1[7:0]};
wire WUR0_C1;
xtdelay1 #(1) iWUR0_C1(.xtin(WUR0_C0), .xtout(WUR0_C1), .clk(clk));
assign gfmod_kill_C1 = (1'b0) & (WUR0_C1);
```

```
endmodule

module xmTIE_ref (
TIE_inst_R,
TIE_asRead_R,
TIE_atRead_R,
TIE_atWrite_R,
TIE_arWrite_R,
TIE_asWrite_R,
TIE_aWriteM_R,
TIE_aDataKill_E,
TIE_aWriteData_E,
TIE_aDataKill_M,
TIE_aWriteData_M,
TIE_Load_R,
TIE_Store_R,
TIE_LSSize_R,
TIE_LSIndexed_R,
TIE_LSOffset_R,
TIE_MemLoadData_M,
TIE_MemStoreData8_E,
TIE_MemStoreData16_E,
TIE_MemStoreData32_E,
TIE_MemStoreData64_E,
TIE_MemStoreData128_E,
TIE_Stall_R,
TIE_Exception_E,
TIE_ExcCause_E,
TIE_bsRead_R,
TIE_btRead_R,
TIE_btWrite_R,
TIE_brWrite_R,
TIE_bsWrite_R,
TIE_bsReadSize_R,
TIE_btReadSize_R,
TIE_bWriteSize_R,
TIE_bsReadData_E,
TIE_btReadData_E,
TIE_bWriteData1_E,
TIE_bWriteData2_E,
TIE_bWriteData4_E,
TIE_bWriteData8_E,
TIE_bWriteData16_E,
TIE_bDataKill_E,
CPEnable,
Instr_R,
SBus_E,
TBus_E,
MemOpAddr_E,
Kill_E,
Except_W,
Replay_W,
G1WCLK,
Reset
);
output TIE_inst_R;
output TIE_asRead_R;
```

```
output TIE_atRead_R;
output TIE_atWrite_R;
output TIE_arWrite_R;
output TIE_asWrite_R;
output TIE_aWriteM_R;
output TIE_aDataKill_E;
output [31:0] TIE_aWriteData_E;
output TIE_aDataKill_M;
output [31:0] TIE_aWriteData_M;
output TIE_Load_R;
output TIE_Store_R;
output [4:0] TIE_LSSize_R;
output TIE_LSIndexed_R;
output [31:0] TIE_LSOffset_R;
input [127:0] TIE_MemLoadData_M;
output [7:0] TIE_MemStoreData8_E;
output [15:0] TIE_MemStoreData16_E;
output [31:0] TIE_MemStoreData32_E;
output [63:0] TIE_MemStoreData64_E;
output [127:0] TIE_MemStoreData128_E;
output TIE_Stall_R;
output TIE_Exception_E;
output [5:0] TIE_ExcCause_E;
output TIE_bsRead_R;
output TIE_btRead_R;
output TIE_btWrite_R;
output TIE_brWrite_R;
output TIE_bsWrite_R;
output [4:0] TIE_bsReadSize_R;
output [4:0] TIE_btReadSize_R;
output [4:0] TIE_bWriteSize_R;
input [15:0] TIE_bsReadData_E;
input [15:0] TIE_btReadData_E;
output TIE_bWriteData1_E;
output [1:0] TIE_bWriteData2_E;
output [3:0] TIE_bWriteData4_E;
output [7:0] TIE_bWriteData8_E;
output [15:0] TIE_bWriteData16_E;
output TIE_bDataKill_E;
input [7:0] CPEnable;
input [23:0] Instr_R;
input [31:0] SBus_E;
input [31:0] TBus_E;
input [31:0] MemOpAddr_E;
input Kill_E;
input Except_W;
input Replay_W;
input G1WCLK;
input Reset;

// unused signals
wire TMode = 0;

// control signals
wire KillPipe_W;
wire clk;
```

```
// decoded signals
wire GFADD8_C0;
wire GFADD8I_C0;
wire GFMULX8_C0;
wire GFRWMOD8_C0;
wire LGF8_I_C0;
wire SGF8_I_C0;
wire LGF8_IU_C0;
wire SGF8_IU_C0;
wire LGF8_X_C0;
wire SGF8_X_C0;
wire LGF8_XU_C0;
wire SGF8_XU_C0;
wire RUR0_C0;
wire WUR0_C0;
wire [31:0] imm4_C0;
wire [7:0] imm8_C0;
wire art_use_C0;
wire art_def_C0;
wire ars_use_C0;
wire ars_def_C0;
wire arr_use_C0;
wire arr_def_C0;
wire br_use_C0;
wire br_def_C0;
wire bs_use_C0;
wire bs_def_C0;
wire bt_use_C0;
wire bt_def_C0;
wire bs4_use_C0;
wire bs4_def_C0;
wire bs8_use_C0;
wire bs8_def_C0;
wire gr_use_C0;
wire gr_def_C0;
wire gs_use_C0;
wire gs_def_C0;
wire gt_use_C0;
wire gt_def_C0;
wire gfmod_use1_C0;
wire gfmod_def1_C0;
wire AR_rd0_use1_C0;
wire AR_rd0_width32_C0;
wire AR_rd1_use1_C0;
wire AR_rd1_width32_C0;
wire AR_wd_def1_C0;
wire AR_wd_width32_C0;
wire [3:0] gf_rd0_addr_C0;
wire gf_rd0_use1_C0;
wire gf_rd0_width8_C0;
wire [3:0] gf_rd1_addr_C0;
wire gf_rd1_use1_C0;
wire gf_rd1_width8_C0;
wire [3:0] gf_rd2_addr_C0;
wire gf_rd2_use1_C0;
wire gf_rd2_width8_C0;
wire [3:0] gf_wd_addr_C0;
```

```
wire gf_wd_def2_C0;
wire gf_wd_def1_C0;
wire gf_wd_width8_C0;
wire GFADD8_semantic_C0;
wire GFADD8I_semantic_C0;
wire GFMULX8_semantic_C0;
wire GFRWMOD8_semantic_C0;
wire LGF8_I_semantic_C0;
wire LGF8_IU_semantic_C0;
wire LGF8_X_semantic_C0;
wire LGF8_XU_semantic_C0;
wire SGF8_I_semantic_C0;
wire SGF8_IU_semantic_C0;
wire SGF8_X_semantic_C0;
wire SGF8_XU_semantic_C0;
wire RUR0_semantic_C0;
wire WUR0_semantic_C0;
wire load_instruction_C0;
wire store_instruction_C0;
wire TIE_Inst_C0;
wire [23:0] Inst_C0;

// state data, write-enable and stall signals
wire [7:0] gfmod_ps_C1;
wire [7:0] gfmod_ns_C1;
wire gfmod_kill_C1;
wire gfmod_Stall_C1;

// register data, write-enable and stall signals
wire [31:0] AR_rd0_data_C1;
wire [31:0] AR_rd1_data_C1;
wire [31:0] AR_wd_data32_C1;
wire AR_wd_kill_C1;
wire [7:0] gf_rd0_data_C1;
wire [7:0] gf_rd1_data_C1;
wire [7:0] gf_rd2_data_C1;
wire [7:0] gf_wd_data8_C2;
wire gf_wd_kill_C2;
wire [7:0] gf_wd_data8_C1;
wire gf_wd_kill_C1;
wire gf_Stall_C1;

// operands
wire [31:0] art_i_C1;
wire [31:0] art_o_C1;
wire art_kill_C1;
wire [31:0] ars_i_C1;
wire [31:0] ars_o_C1;
wire ars_kill_C1;
wire [31:0] arr_o_C1;
wire arr_kill_C1;
wire [7:0] gr_i_C1;
wire [7:0] gr_o_C2;
wire gr_kill_C2;
wire [7:0] gr_o_C1;
wire gr_kill_C1;
wire [7:0] gs_i_C1;
```

```
wire [7:0] gt_i_C1;
wire [7:0] gt_o_C2;
wire gt_kill_C2;
wire [7:0] gt_o_C1;
wire gt_kill_C1;

// output state of semantic GFADD8

// output interface of semantic GFADD8

// output operand of semantic GFADD8
wire [7:0] GFADD8_gr_o_C1;
wire GFADD8_gr_kill_C1;

// output state of semantic GFADD8I

// output interface of semantic GFADD8I

// output operand of semantic GFADD8I
wire [7:0] GFADD8I_gr_o_C1;
wire GFADD8I_gr_kill_C1;

// output state of semantic GFMULX8

// output interface of semantic GFMULX8

// output operand of semantic GFMULX8
wire [7:0] GFMULX8_gr_o_C1;
wire GFMULX8_gr_kill_C1;

// output state of semantic GFRWMOD8
wire [7:0] GFRWMOD8_gfmod_ns_C1;
wire GFRWMOD8_gfmod_kill_C1;

// output interface of semantic GFRWMOD8

// output operand of semantic GFRWMOD8
wire [7:0] GFRWMOD8_gt_o_C1;
wire GFRWMOD8_gt_kill_C1;

// output state of semantic LGF8_I

// output interface of semantic LGF8_I
wire [4:0] LGF8_I_LSSize_C0;
wire [31:0] LGF8_I_VAddrBase_C1;
wire [31:0] LGF8_I_VAddrOffset_C0;
wire LGF8_I_LSIndexed_C0;

// output operand of semantic LGF8_I
wire [7:0] LGF8_I_gt_o_C2;
wire LGF8_I_gt_kill_C2;

// output state of semantic LGF8_IU

// output interface of semantic LGF8_IU
wire [4:0] LGF8_IU_LSSize_C0;
wire [31:0] LGF8_IU_VAddrBase_C1;
```

```
wire [31:0] LGF8_IU_VAddrOffset_C0;
wire LGF8_IU_LSIndexed_C0;

// output operand of semantic LGF8_IU
wire [7:0] LGF8_IU_gt_o_C2;
wire LGF8_IU_gt_kill_C2;
wire [31:0] LGF8_IU_ars_o_C1;
wire LGF8_IU_ars_kill_C1;

// output state of semantic LGF8_X

// output interface of semantic LGF8_X
wire [4:0] LGF8_X_LSSize_C0;
wire [31:0] LGF8_X_VAddrBase_C1;
wire [31:0] LGF8_X_VAddrIndex_C1;
wire LGF8_X_LSIndexed_C0;

// output operand of semantic LGF8_X
wire [7:0] LGF8_X_gr_o_C2;
wire LGF8_X_gr_kill_C2;

// output state of semantic LGF8_XU

// output interface of semantic LGF8_XU
wire [4:0] LGF8_XU_LSSize_C0;
wire [31:0] LGF8_XU_VAddrBase_C1;
wire [31:0] LGF8_XU_VAddrIndex_C1;
wire LGF8_XU_LSIndexed_C0;

// output operand of semantic LGF8_XU
wire [7:0] LGF8_XU_gr_o_C2;
wire LGF8_XU_gr_kill_C2;
wire [31:0] LGF8_XU_ars_o_C1;
wire LGF8_XU_ars_kill_C1;

// output state of semantic SGF8_I

// output interface of semantic SGF8_I
wire [4:0] SGF8_I_LSSize_C0;
wire [7:0] SGF8_I_MemDataOut8_C1;
wire [31:0] SGF8_I_VAddrBase_C1;
wire [31:0] SGF8_I_VAddrOffset_C0;
wire SGF8_I_LSIndexed_C0;

// output operand of semantic SGF8_I

// output state of semantic SGF8_IU

// output interface of semantic SGF8_IU
wire [4:0] SGF8_IU_LSSize_C0;
wire [7:0] SGF8_IU_MemDataOut8_C1;
wire [31:0] SGF8_IU_VAddrBase_C1;
wire [31:0] SGF8_IU_VAddrOffset_C0;
wire SGF8_IU_LSIndexed_C0;

// output operand of semantic SGF8_IU
wire [31:0] SGF8_IU_ars_o_C1;
```

```
wire SGF8_IU_ars_kill_C1;

// output state of semantic SGF8_X

// output interface of semantic SGF8_X
wire [4:0] SGF8_X_LSSize_C0;
wire [7:0] SGF8_X_MemDataOut8_C1;
wire [31:0] SGF8_X_VAddrBase_C1;
wire [31:0] SGF8_X_VAddrIndex_C1;
wire SGF8_X_LSIndexed_C0;

// output operand of semantic SGF8_X

// output state of semantic SGF8_XU

// output interface of semantic SGF8_XU
wire [4:0] SGF8_XU_LSSize_C0;
wire [7:0] SGF8_XU_MemDataOut8_C1;
wire [31:0] SGF8_XU_VAddrBase_C1;
wire [31:0] SGF8_XU_VAddrIndex_C1;
wire SGF8_XU_LSIndexed_C0;

// output operand of semantic SGF8_XU
wire [31:0] SGF8_XU_ars_o_C1;
wire SGF8_XU_ars_kill_C1;

// output state of semantic RUR0

// output interface of semantic RUR0

// output operand of semantic RUR0
wire [31:0] RUR0_arr_o_C1;
wire RUR0_arr_kill_C1;

// output state of semantic WUR0
wire [7:0] WUR0_gfmod_ns_C1;
wire WUR0_gfmod_kill_C1;

// output interface of semantic WUR0

// output operand of semantic WUR0

// TIE-defined interface signals
wire [31:0] VAddr_C1;
wire [31:0] VAddrBase_C1;
wire [31:0] VAddrOffset_C0;
wire [31:0] VAddrIndex_C1;
wire [31:0] VAddrIn_C1;
wire [4:0] LSSize_C0;
wire LSIndexed_C0;
wire [127:0] MemDataIn128_C2;
wire [63:0] MemDataIn64_C2;
wire [31:0] MemDataIn32_C2;
wire [15:0] MemDataIn16_C2;
wire [7:0] MemDataIn8_C2;
wire [127:0] MemDataOut128_C1;
wire [63:0] MemDataOut64_C1;
```

```
wire [31:0] MemDataOut32_C1;
wire [15:0] MemDataOut16_C1;
wire [7:0] MemDataOut8_C1;
wire Exception_C1;
wire [5:0] ExcCause_C1;
wire [7:0] CPEnable_C1;
    xtflop #(1) reset(localReset, Reset, G1WCLK);

xmTIE_decoder TIE_decoder (
    .GFADD8(GFADD8_C0),
    .GFADD8I(GFADD8I_C0),
    .GFMULX8(GFMULX8_C0),
    .GFRWMOD8(GFRWMOD8_C0),
    .LGF8_I(LGF8_I_C0),
    .SGF8_I(SGF8_I_C0),
    .LGF8_IU(LGF8_IU_C0),
    .SGF8_IU(SGF8_IU_C0),
    .LGF8_X(LGF8_X_C0),
    .SGF8_X(SGF8_X_C0),
    .LGF8_XU(LGF8_XU_C0),
    .SGF8_XU(SGF8_XU_C0),
    .RUR0(RUR0_C0),
    .WUR0(WUR0_C0),
    .imm4(imm4_C0),
    .imm8(imm8_C0),
    .art_use(art_use_C0),
    .art_def(art_def_C0),
    .ars_use(ars_use_C0),
    .ars_def(ars_def_C0),
    .arr_use(arr_use_C0),
    .arr_def(arr_def_C0),
    .br_use(br_use_C0),
    .br_def(br_def_C0),
    .bs_use(bs_use_C0),
    .bs_def(bs_def_C0),
    .bt_use(bt_use_C0),
    .bt_def(bt_def_C0),
    .bs4_use(bs4_use_C0),
    .bs4_def(bs4_def_C0),
    .bs8_use(bs8_use_C0),
    .bs8_def(bs8_def_C0),
    .gr_use(gr_use_C0),
    .gr_def(gr_def_C0),
    .gs_use(gs_use_C0),
    .gs_def(gs_def_C0),
    .gt_use(gt_use_C0),
    .gt_def(gt_def_C0),
    .gfmod_use1(gfmod_use1_C0),
    .gfmod_def1(gfmod_def1_C0),
    .AR_rd0_use1(AR_rd0_use1_C0),
    .AR_rd0_width32(AR_rd0_width32_C0),
    .AR_rd1_use1(AR_rd1_use1_C0),
    .AR_rd1_width32(AR_rd1_width32_C0),
    .AR_wd_def1(AR_wd_def1_C0),
    .AR_wd_width32(AR_wd_width32_C0),
    .gf_rd0_addr(gf_rd0_addr_C0),
    .gf_rd0_use1(gf_rd0_use1_C0),
```

```
        .gf_rd0_width8(gf_rd0_width8_C0),
        .gf_rd1_addr(gf_rd1_addr_C0),
        .gf_rd1_use1(gf_rd1_use1_C0),
        .gf_rd1_width8(gf_rd1_width8_C0),
        .gf_rd2_addr(gf_rd2_addr_C0),
        .gf_rd2_use1(gf_rd2_use1_C0),
        .gf_rd2_width8(gf_rd2_width8_C0),
        .gf_wd_addr(gf_wd_addr_C0),
        .gf_wd_def2(gf_wd_def2_C0),
        .gf_wd_def1(gf_wd_def1_C0),
        .gf_wd_width8(gf_wd_width8_C0),
        .GFADD8_semantic(GFADD8_semantic_C0),
        .GFADD8I_semantic(GFADD8I_semantic_C0),
        .GFMULX8_semantic(GFMULX8_semantic_C0),
        .GFRWMOD8_semantic(GFRWMOD8_semantic_C0),
        .LGF8_I_semantic(LGF8_I_semantic_C0),
        .LGF8_IU_semantic(LGF8_IU_semantic_C0),
        .LGF8_X_semantic(LGF8_X_semantic_C0),
        .LGF8_XU_semantic(LGF8_XU_semantic_C0),
        .SGF8_I_semantic(SGF8_I_semantic_C0),
        .SGF8_IU_semantic(SGF8_IU_semantic_C0),
        .SGF8_X_semantic(SGF8_X_semantic_C0),
        .SGF8_XU_semantic(SGF8_XU_semantic_C0),
        .RUR0_semantic(RUR0_semantic_C0),
        .WUR0_semantic(WUR0_semantic_C0),
        .load_instruction(load_instruction_C0),
        .store_instruction(store_instruction_C0),
        .TIE_Inst(TIE_Inst_C0),
        .Inst(Inst_C0)
    );

    xmTIE_GFADD8 TIE_GFADD8(
        .GFADD8_C0(GFADD8_C0),
        .gr_o_C1(GFADD8_gr_o_C1),
        .gr_kill_C1(GFADD8_gr_kill_C1),
        .gs_i_C1(gs_i_C1),
        .gt_i_C1(gt_i_C1),
        .clk(clk));

    xmTIE_GFADD8I TIE_GFADD8I(
        .GFADD8I_C0(GFADD8I_C0),
        .gr_o_C1(GFADD8I_gr_o_C1),
        .gr_kill_C1(GFADD8I_gr_kill_C1),
        .gs_i_C1(gs_i_C1),
        .imm4_C0(imm4_C0),
        .clk(clk));

    xmTIE_GFMULX8 TIE_GFMULX8(
        .GFMULX8_C0(GFMULX8_C0),
        .gr_o_C1(GFMULX8_gr_o_C1),
        .gr_kill_C1(GFMULX8_gr_kill_C1),
        .gs_i_C1(gs_i_C1),
        .gfmod_ps_C1(gfmod_ps_C1),
        .clk(clk));

    xmTIE_GFRWMOD8 TIE_GFRWMOD8(
        .GFRWMOD8_C0(GFRWMOD8_C0),
```

```
    .gt_i_C1(gt_i_C1),
    .gt_o_C1(GFRWMOD8_gt_o_C1),
    .gt_kill_C1(GFRWMOD8_gt_kill_C1),
    .gfmod_ps_C1(gfmod_ps_C1),
    .gfmod_ns_C1(GFRWMOD8_gfmod_ns_C1),
    .gfmod_kill_C1(GFRWMOD8_gfmod_kill_C1),
    .clk(clk));

xmTIE_LGF8_I TIE_LGF8_I(
    .LGF8_I_C0(LGF8_I_C0),
    .gt_o_C2(LGF8_I_gt_o_C2),
    .gt_kill_C2(LGF8_I_gt_kill_C2),
    .ars_i_C1(ars_i_C1),
    .imm8_C0(imm8_C0),
    .MemDataIn8_C2(MemDataIn8_C2),
    .LSSize_C0(LGF8_I_LSSize_C0),
    .VAddrBase_C1(LGF8_I_VAddrBase_C1),
    .VAddrOffset_C0(LGF8_I_VAddrOffset_C0),
    .LSIndexed_C0(LGF8_I_LSIndexed_C0),
    .clk(clk));

xmTIE_LGF8_IU TIE_LGF8_IU(
    .LGF8_IU_C0(LGF8_IU_C0),
    .gt_o_C2(LGF8_IU_gt_o_C2),
    .gt_kill_C2(LGF8_IU_gt_kill_C2),
    .ars_i_C1(ars_i_C1),
    .ars_o_C1(LGF8_IU_ars_o_C1),
    .ars_kill_C1(LGF8_IU_ars_kill_C1),
    .imm8_C0(imm8_C0),
    .MemDataIn8_C2(MemDataIn8_C2),
    .VAddrIn_C1(VAddrIn_C1),
    .LSSize_C0(LGF8_IU_LSSize_C0),
    .VAddrBase_C1(LGF8_IU_VAddrBase_C1),
    .VAddrOffset_C0(LGF8_IU_VAddrOffset_C0),
    .LSIndexed_C0(LGF8_IU_LSIndexed_C0),
    .clk(clk));

xmTIE_LGF8_X TIE_LGF8_X(
    .LGF8_X_C0(LGF8_X_C0),
    .gr_o_C2(LGF8_X_gr_o_C2),
    .gr_kill_C2(LGF8_X_gr_kill_C2),
    .ars_i_C1(ars_i_C1),
    .art_i_C1(art_i_C1),
    .MemDataIn8_C2(MemDataIn8_C2),
    .VAddrIn_C1(VAddrIn_C1),
    .LSSize_C0(LGF8_X_LSSize_C0),
    .VAddrBase_C1(LGF8_X_VAddrBase_C1),
    .VAddrIndex_C1(LGF8_X_VAddrIndex_C1),
    .LSIndexed_C0(LGF8_X_LSIndexed_C0),
    .clk(clk));

xmTIE_LGF8_XU TIE_LGF8_XU(
    .LGF8_XU_C0(LGF8_XU_C0),
    .gr_o_C2(LGF8_XU_gr_o_C2),
    .gr_kill_C2(LGF8_XU_gr_kill_C2),
    .ars_i_C1(ars_i_C1),
    .ars_o_C1(LGF8_XU_ars_o_C1),
```

```
        .ars_kill_C1(LGF8_XU_ars_kill_C1),
        .art_i_C1(art_i_C1),
        .MemDataIn8_C2(MemDataIn8_C2),
        .VAddrIn_C1(VAddrIn_C1),
        .LSSize_C0(LGF8_XU_LSSize_C0),
        .VAddrBase_C1(LGF8_XU_VAddrBase_C1),
        .VAddrIndex_C1(LGF8_XU_VAddrIndex_C1),
        .LSIndexed_C0(LGF8_XU_LSIndexed_C0),
        .clk(clk));

    xmTIE_SGF8_I TIE_SGF8_I(
        .SGF8_I_C0(SGF8_I_C0),
        .gt_i_C1(gt_i_C1),
        .ars_i_C1(ars_i_C1),
        .imm8_C0(imm8_C0),
        .LSSize_C0(SGF8_I_LSSize_C0),
        .MemDataOut8_C1(SGF8_I_MemDataOut8_C1),
        .VAddrBase_C1(SGF8_I_VAddrBase_C1),
        .VAddrOffset_C0(SGF8_I_VAddrOffset_C0),
        .LSIndexed_C0(SGF8_I_LSIndexed_C0),
        .clk(clk));

    xmTIE_SGF8_IU TIE_SGF8_IU(
        .SGF8_IU_C0(SGF8_IU_C0),
        .gt_i_C1(gt_i_C1),
        .ars_i_C1(ars_i_C1),
        .ars_o_C1(SGF8_IU_ars_o_C1),
        .ars_kill_C1(SGF8_IU_ars_kill_C1),
        .imm8_C0(imm8_C0),
        .VAddrIn_C1(VAddrIn_C1),
        .LSSize_C0(SGF8_IU_LSSize_C0),
        .MemDataOut8_C1(SGF8_IU_MemDataOut8_C1),
        .VAddrBase_C1(SGF8_IU_VAddrBase_C1),
        .VAddrOffset_C0(SGF8_IU_VAddrOffset_C0),
        .LSIndexed_C0(SGF8_IU_LSIndexed_C0),
        .clk(clk));

    xmTIE_SGF8_X TIE_SGF8_X(
        .SGF8_X_C0(SGF8_X_C0),
        .gr_i_C1(gr_i_C1),
        .ars_i_C1(ars_i_C1),
        .art_i_C1(art_i_C1),
        .LSSize_C0(SGF8_X_LSSize_C0),
        .MemDataOut8_C1(SGF8_X_MemDataOut8_C1),
        .VAddrBase_C1(SGF8_X_VAddrBase_C1),
        .VAddrIndex_C1(SGF8_X_VAddrIndex_C1),
        .LSIndexed_C0(SGF8_X_LSIndexed_C0),
        .clk(clk));

    xmTIE_SGF8_XU TIE_SGF8_XU(
        .SGF8_XU_C0(SGF8_XU_C0),
        .gr_i_C1(gr_i_C1),
        .ars_i_C1(ars_i_C1),
        .ars_o_C1(SGF8_XU_ars_o_C1),
        .ars_kill_C1(SGF8_XU_ars_kill_C1),
        .art_i_C1(art_i_C1),
        .VAddrIn_C1(VAddrIn_C1),
```

```
        .LSSize_C0(SGF8_XU_LSSize_C0),
        .MemDataOut8_C1(SGF8_XU_MemDataOut8_C1),
        .VAddrBase_C1(SGF8_XU_VAddrBase_C1),
        .VAddrIndex_C1(SGF8_XU_VAddrIndex_C1),
        .LSIndexed_C0(SGF8_XU_LSIndexed_C0),
        .clk(clk));

    xmTIE_RUR0 TIE_RUR0(
        .RUR0_C0(RUR0_C0),
        .arr_o_C1(RUR0_arr_o_C1),
        .arr_kill_C1(RUR0_arr_kill_C1),
        .gfmod_ps_C1(gfmod_ps_C1),
        .clk(clk));

    xmTIE_WUR0 TIE_WUR0(
        .WUR0_C0(WUR0_C0),
        .art_i_C1(art_i_C1),
        .gfmod_ns_C1(WUR0_gfmod_ns_C1),
        .gfmod_kill_C1(WUR0_gfmod_kill_C1),
        .clk(clk));

    xmTIE_gfmod_State TIE_gfmod_State (
        .ps_width8_C0(1'b1),
        .ps_use1_C0(gfmod_use1_C0),
        .ps_data_C1(gfmod_ps_C1),
        .ns_width8_C0(1'b1),
        .ns_def1_C0(gfmod_def1_C0),
        .ns_data8_C1(gfmod_ns_C1),
        .ns_wen_C1(~gfmod_kill_C1),
        .Kill_E(Kill_E),
        .KillPipe_W(KillPipe_W),
        .Stall_R(gfmod_Stall_C1),
        .clk(clk)
    );

    xmTIE_gf_Regfile TIE_gf_Regfile (
        .rd0_addr_C0(gf_rd0_addr_C0),
        .rd0_use1_C0(gf_rd0_use1_C0),
        .rd0_data_C1(gf_rd0_data_C1),
        .rd0_width8_C0(gf_rd0_width8_C0),
        .rd1_addr_C0(gf_rd1_addr_C0),
        .rd1_use1_C0(gf_rd1_use1_C0),
        .rd1_data_C1(gf_rd1_data_C1),
        .rd1_width8_C0(gf_rd1_width8_C0),
        .rd2_addr_C0(gf_rd2_addr_C0),
        .rd2_use1_C0(gf_rd2_use1_C0),
        .rd2_data_C1(gf_rd2_data_C1),
        .rd2_width8_C0(gf_rd2_width8_C0),
        .wd_addr_C0(gf_wd_addr_C0),
        .wd_def2_C0(gf_wd_def2_C0),
        .wd_wen_C2(~gf_wd_kill_C2),
        .wd_data8_C2(gf_wd_data8_C2),
        .wd_def1_C0(gf_wd_def1_C0),
        .wd_wen_C1(~gf_wd_kill_C1),
        .wd_data8_C1(gf_wd_data8_C1),
        .wd_width8_C0(gf_wd_width8_C0),
        .Kill_E(Kill_E),
```

```
    .KillPipe_W(KillPipe_W),
    .Stall_R(gf_Stall_C1),
    .clk(clk)
);

// Stall logic
assign TIE_Stall_R = 1'b0
          | gf_Stall_C1
          | gfmod_Stall_C1;

// pipeline semantic select signals to each stage
wire LGF8_I_semantic_C1;
xtdelay1 #(1) iLGF8_I_semantic_C1(.xtin(LGF8_I_semantic_C0),
.xtout(LGF8_I_semantic_C1), .clk(clk));
wire LGF8_IU_semantic_C1;
xtdelay1 #(1) iLGF8_IU_semantic_C1(.xtin(LGF8_IU_semantic_C0),
.xtout(LGF8_IU_semantic_C1), .clk(clk));
wire LGF8_X_semantic_C1;
xtdelay1 #(1) iLGF8_X_semantic_C1(.xtin(LGF8_X_semantic_C0),
.xtout(LGF8_X_semantic_C1), .clk(clk));
wire LGF8_XU_semantic_C1;
xtdelay1 #(1) iLGF8_XU_semantic_C1(.xtin(LGF8_XU_semantic_C0),
.xtout(LGF8_XU_semantic_C1), .clk(clk));
wire SGF8_I_semantic_C1;
xtdelay1 #(1) iSGF8_I_semantic_C1(.xtin(SGF8_I_semantic_C0),
.xtout(SGF8_I_semantic_C1), .clk(clk));
wire SGF8_IU_semantic_C1;
xtdelay1 #(1) iSGF8_IU_semantic_C1(.xtin(SGF8_IU_semantic_C0),
.xtout(SGF8_IU_semantic_C1), .clk(clk));
wire SGF8_X_semantic_C1;
xtdelay1 #(1) iSGF8_X_semantic_C1(.xtin(SGF8_X_semantic_C0),
.xtout(SGF8_X_semantic_C1), .clk(clk));
wire SGF8_XU_semantic_C1;
xtdelay1 #(1) iSGF8_XU_semantic_C1(.xtin(SGF8_XU_semantic_C0),
.xtout(SGF8_XU_semantic_C1), .clk(clk));
wire GFRWMOD8_semantic_C1;
xtdelay1 #(1) iGFRWMOD8_semantic_C1(.xtin(GFRWMOD8_semantic_C0),
.xtout(GFRWMOD8_semantic_C1), .clk(clk));
wire WUR0_semantic_C1;
xtdelay1 #(1) iWUR0_semantic_C1(.xtin(WUR0_semantic_C0),
.xtout(WUR0_semantic_C1), .clk(clk));
wire RUR0_semantic_C1;
xtdelay1 #(1) iRUR0_semantic_C1(.xtin(RUR0_semantic_C0),
.xtout(RUR0_semantic_C1), .clk(clk));
wire LGF8_X_semantic_C2;
xtdelay2 #(1) iLGF8_X_semantic_C2(.xtin(LGF8_X_semantic_C0),
.xtout(LGF8_X_semantic_C2), .clk(clk));
wire LGF8_XU_semantic_C2;
xtdelay2 #(1) iLGF8_XU_semantic_C2(.xtin(LGF8_XU_semantic_C0),
.xtout(LGF8_XU_semantic_C2), .clk(clk));
wire GFADD8_semantic_C1;
xtdelay1 #(1) iGFADD8_semantic_C1(.xtin(GFADD8_semantic_C0),
.xtout(GFADD8_semantic_C1), .clk(clk));
wire GFADD8I_semantic_C1;
xtdelay1 #(1) iGFADD8I_semantic_C1(.xtin(GFADD8I_semantic_C0),
.xtout(GFADD8I_semantic_C1), .clk(clk));
wire GFMULX8_semantic_C1;
```

```
xtdelay1 #(1) iGFMULX8_semantic_C1(.xtin(GFMULX8_semantic_C0),
.xtout(GFMULX8_semantic_C1), .clk(clk));
wire LGF8_I_semantic_C2;
xtdelay2 #(1) iLGF8_I_semantic_C2(.xtin(LGF8_I_semantic_C0),
.xtout(LGF8_I_semantic_C2), .clk(clk));
wire LGF8_IU_semantic_C2;
xtdelay2 #(1) iLGF8_IU_semantic_C2(.xtin(LGF8_IU_semantic_C0),
.xtout(LGF8_IU_semantic_C2), .clk(clk));

// combine output interface signals from all semantics
assign VAddr_C1 = 32'b0;
assign VAddrBase_C1 = 32'b0
          | (LGF8_I_VAddrBase_C1 & {32{LGF8_I_semantic_C1}})
          | (LGF8_IU_VAddrBase_C1 & {32{LGF8_IU_semantic_C1}})
          | (LGF8_X_VAddrBase_C1 & {32{LGF8_X_semantic_C1}})
          | (LGF8_XU_VAddrBase_C1 & {32{LGF8_XU_semantic_C1}})
          | (SGF8_I_VAddrBase_C1 & {32{SGF8_I_semantic_C1}})
          | (SGF8_IU_VAddrBase_C1 & {32{SGF8_IU_semantic_C1}})
          | (SGF8_X_VAddrBase_C1 & {32{SGF8_X_semantic_C1}})
          | (SGF8_XU_VAddrBase_C1 & {32{SGF8_XU_semantic_C1}});
assign VAddrOffset_C0 = 32'b0
          | (LGF8_I_VAddrOffset_C0 & {32{LGF8_I_semantic_C0}})
          | (LGF8_IU_VAddrOffset_C0 & {32{LGF8_IU_semantic_C0}})
          | (SGF8_I_VAddrOffset_C0 & {32{SGF8_I_semantic_C0}})
          | (SGF8_IU_VAddrOffset_C0 & {32{SGF8_IU_semantic_C0}});
assign VAddrIndex_C1 = 32'b0
          | (LGF8_X_VAddrIndex_C1 & {32{LGF8_X_semantic_C1}})
          | (LGF8_XU_VAddrIndex_C1 & {32{LGF8_XU_semantic_C1}})
          | (SGF8_X_VAddrIndex_C1 & {32{SGF8_X_semantic_C1}})
          | (SGF8_XU_VAddrIndex_C1 & {32{SGF8_XU_semantic_C1}});
assign LSSize_C0 = 5'b0
          | (LGF8_I_LSSize_C0 & {5{LGF8_I_semantic_C0}})
          | (LGF8_IU_LSSize_C0 & {5{LGF8_IU_semantic_C0}})
          | (LGF8_X_LSSize_C0 & {5{LGF8_X_semantic_C0}})
          | (LGF8_XU_LSSize_C0 & {5{LGF8_XU_semantic_C0}})
          | (SGF8_I_LSSize_C0 & {5{SGF8_I_semantic_C0}})
          | (SGF8_IU_LSSize_C0 & {5{SGF8_IU_semantic_C0}})
          | (SGF8_X_LSSize_C0 & {5{SGF8_X_semantic_C0}})
          | (SGF8_XU_LSSize_C0 & {5{SGF8_XU_semantic_C0}});
assign LSIndexed_C0 = 1'b0
          | (LGF8_I_LSIndexed_C0 & LGF8_I_semantic_C0)
          | (LGF8_IU_LSIndexed_C0 & LGF8_IU_semantic_C0)
          | (LGF8_X_LSIndexed_C0 & LGF8_X_semantic_C0)
          | (LGF8_XU_LSIndexed_C0 & LGF8_XU_semantic_C0)
          | (SGF8_I_LSIndexed_C0 & SGF8_I_semantic_C0)
          | (SGF8_IU_LSIndexed_C0 & SGF8_IU_semantic_C0)
          | (SGF8_X_LSIndexed_C0 & SGF8_X_semantic_C0)
          | (SGF8_XU_LSIndexed_C0 & SGF8_XU_semantic_C0);
assign MemDataOut128_C1 = 128'b0;
assign MemDataOut64_C1 = 64'b0;
assign MemDataOut32_C1 = 32'b0;
assign MemDataOut16_C1 = 16'b0;
assign MemDataOut8_C1 = 8'b0
          | (SGF8_I_MemDataOut8_C1 & {8{SGF8_I_semantic_C1}})
          | (SGF8_IU_MemDataOut8_C1 & {8{SGF8_IU_semantic_C1}})
          | (SGF8_X_MemDataOut8_C1 & {8{SGF8_X_semantic_C1}})
          | (SGF8_XU_MemDataOut8_C1 & {8{SGF8_XU_semantic_C1}});
```

```
assign Exception_C1 = 1'b0;
assign ExcCause_C1 = 6'b0;

// combine output state signals from all semantics
assign gfmod_ns_C1 = 8'b0
          | (GFRWMOD8_gfmod_ns_C1 & {8{GFRWMOD8_semantic_C1}})
          | (WUR0_gfmod_ns_C1 & {8{WUR0_semantic_C1}});
assign gfmod_kill_C1 = 1'b0
          | (GFRWMOD8_gfmod_kill_C1 & GFRWMOD8_semantic_C1)
          | (WUR0_gfmod_kill_C1 & WUR0_semantic_C1);

// combine output operand signals from all semantics
assign art_o_C1 = 32'b0;
assign art_kill_C1 = 1'b0;
assign ars_o_C1 = 32'b0
          | (LGF8_IU_ars_o_C1 & {32{LGF8_IU_semantic_C1}})
          | (LGF8_XU_ars_o_C1 & {32{LGF8_XU_semantic_C1}})
          | (SGF8_IU_ars_o_C1 & {32{SGF8_IU_semantic_C1}})
          | (SGF8_XU_ars_o_C1 & {32{SGF8_XU_semantic_C1}});
assign ars_kill_C1 = 1'b0
          | (LGF8_IU_ars_kill_C1 & LGF8_IU_semantic_C1)
          | (LGF8_XU_ars_kill_C1 & LGF8_XU_semantic_C1)
          | (SGF8_IU_ars_kill_C1 & SGF8_IU_semantic_C1)
          | (SGF8_XU_ars_kill_C1 & SGF8_XU_semantic_C1);
assign arr_o_C1 = 32'b0
          | (RUR0_arr_o_C1 & {32{RUR0_semantic_C1}});
assign arr_kill_C1 = 1'b0
          | (RUR0_arr_kill_C1 & RUR0_semantic_C1);
assign gr_o_C2 = 8'b0
          | (LGF8_X_gr_o_C2 & {8{LGF8_X_semantic_C2}})
          | (LGF8_XU_gr_o_C2 & {8{LGF8_XU_semantic_C2}});
assign gr_kill_C2 = 1'b0
          | (LGF8_X_gr_kill_C2 & LGF8_X_semantic_C2)
          | (LGF8_XU_gr_kill_C2 & LGF8_XU_semantic_C2);
assign gr_o_C1 = 8'b0
          | (GFADD8_gr_o_C1 & {8{GFADD8_semantic_C1}})
          | (GFADD8I_gr_o_C1 & {8{GFADD8I_semantic_C1}})
          | (GFMULX8_gr_o_C1 & {8{GFMULX8_semantic_C1}});
assign gr_kill_C1 = 1'b0
          | (GFADD8_gr_kill_C1 & GFADD8_semantic_C1)
          | (GFADD8I_gr_kill_C1 & GFADD8I_semantic_C1)
          | (GFMULX8_gr_kill_C1 & GFMULX8_semantic_C1);
assign gt_o_C2 = 8'b0
          | (LGF8_I_gt_o_C2 & {8{LGF8_I_semantic_C2}})
          | (LGF8_IU_gt_o_C2 & {8{LGF8_IU_semantic_C2}});
assign gt_kill_C2 = 1'b0
          | (LGF8_I_gt_kill_C2 & LGF8_I_semantic_C2)
          | (LGF8_IU_gt_kill_C2 & LGF8_IU_semantic_C2);
assign gt_o_C1 = 8'b0
          | (GFRWMOD8_gt_o_C1 & {8{GFRWMOD8_semantic_C1}});
assign gt_kill_C1 = 1'b0
          | (GFRWMOD8_gt_kill_C1 & GFRWMOD8_semantic_C1);

// output operand to write port mapping logic
assign AR_wd_data32_C1 = ars_o_C1 | arr_o_C1 | 32'b0;
assign AR_wd_kill_C1 = ars_kill_C1 | arr_kill_C1 | 1'b0;
assign gf_wd_data8_C2 = gt_o_C2 | gr_o_C2 | 8'b0;
```

```
assign gf_wd_kill_C2 = gt_kill_C2 | gr_kill_C2 | 1'b0;
assign gf_wd_data8_C1 = gr_o_C1 | gt_o_C1 | 8'b0;
assign gf_wd_kill_C1 = gr_kill_C1 | gt_kill_C1 | 1'b0;

// read port to input operand mapping logic
assign ars_i_C1 = AR_rd0_data_C1;
assign art_i_C1 = AR_rd1_data_C1;
assign gs_i_C1 = gf_rd0_data_C1;
assign gt_i_C1 = gf_rd1_data_C1;
assign gr_i_C1 = gf_rd2_data_C1;

// logic to support verification
wire ignore_TIE_aWriteData_E = ~(AR_wd_def1_C0 & (TIE_arWrite_R | TIE_asWrite_R
| TIE_atWrite_R) & ~TIE_aDataKill_E);
wire ignore_TIE_aWriteData_M = ~(1'b0 & (TIE_arWrite_R | TIE_asWrite_R |
TIE_atWrite_R) & ~TIE_aDataKill_M);
wire ignore_TIE_bWriteData_E = (~TIE_btWrite_R & ~TIE_btWrite_R) |
TIE_bDataKill_E;
wire ignore_TIE_bWriteData16_E = ignore_TIE_bWriteData_E;
wire ignore_TIE_bWriteData8_E = ignore_TIE_bWriteData_E;
wire ignore_TIE_bWriteData4_E = ignore_TIE_bWriteData_E;
wire ignore_TIE_bWriteData2_E = ignore_TIE_bWriteData_E;
wire ignore_TIE_bWriteData1_E = ignore_TIE_bWriteData_E;
wire ignore_TIE_LSSize_R = ~TIE_Load_R & ~TIE_Store_R;
wire ignore_TIE_LSIndexed_R = ~TIE_Load_R & ~TIE_Store_R;
wire ignore_TIE_LSOffset_R = ~TIE_Load_R & ~TIE_Store_R | TIE_LSIndexed_R;
wire ignore_TIE_MemStoreData128_E = (TIE_LSSize_R != 5'b10000) | ~TIE_Store_R;
wire ignore_TIE_MemStoreData64_E = (TIE_LSSize_R != 5'b01000) | ~TIE_Store_R;
wire ignore_TIE_MemStoreData32_E = (TIE_LSSize_R != 5'b00100) | ~TIE_Store_R;
wire ignore_TIE_MemStoreData16_E = (TIE_LSSize_R != 5'b00010) | ~TIE_Store_R;
wire ignore_TIE_MemStoreData8_E = (TIE_LSSize_R != 5'b00001) | ~TIE_Store_R;

// clock and instructions
assign clk = G1WCLK;
assign Inst_C0 = Instr_R;
assign TIE_inst_R = TIE_Inst_C0;

// AR-related signals to/from core
assign TIE_asRead_R = ars_use_C0;
assign TIE_atRead_R = art_use_C0;
assign TIE_atWrite_R = art_def_C0;
assign TIE_arWrite_R = arr_def_C0;
assign TIE_asWrite_R = ars_def_C0;
assign TIE_aWriteM_R = 0;
assign TIE_aWriteData_E = ignore_TIE_aWriteData_E ? 0 : AR_wd_data32_C1;
assign TIE_aWriteData_M = ignore_TIE_aWriteData_M ? 0 : 0;
assign TIE_aDataKill_E = AR_wd_kill_C1;
assign TIE_aDataKill_M = 0;
assign AR_rd0_data_C1 = SBus_E;
assign AR_rd1_data_C1 = TBus_E;

// BR-related signals to/from core
assign TIE_bsRead_R = 1'b0 | bs_use_C0 | bs4_use_C0 | bs8_use_C0;
assign TIE_btRead_R = 1'b0 | bt_use_C0;
assign TIE_btWrite_R = 1'b0 | bt_def_C0;
assign TIE_bsWrite_R = 1'b0 | bs_def_C0 | bs4_def_C0 | bs8_def_C0;
assign TIE_brWrite_R = 1'b0 | br_def_C0;
```

```
assign TIE_bWriteData16_E = ignore_TIE_bWriteData16_E ? 0 : 0;
assign TIE_bWriteData8_E = ignore_TIE_bWriteData8_E ? 0 : 0;
assign TIE_bWriteData4_E = ignore_TIE_bWriteData4_E ? 0 : 0;
assign TIE_bWriteData2_E = ignore_TIE_bWriteData2_E ? 0 : 0;
assign TIE_bWriteData1_E = ignore_TIE_bWriteData1_E ? 0 : 0;
assign TIE_bDataKill_E = 0;
assign TIE_bWriteSize_R = {1'b0, 1'b0, 1'b0, 1'b0, 1'b0};
assign TIE_bsReadSize_R = {1'b0, 1'b0, 1'b0, 1'b0, 1'b0};
assign TIE_btReadSize_R = {1'b0, 1'b0, 1'b0, 1'b0, 1'b0};

// Load/store signals to/from core
assign TIE_Load_R = load_instruction_C0;
assign TIE_Store_R = store_instruction_C0;
assign TIE_LSSize_R = ignore_TIE_LSSize_R ? 0 : LSSize_C0;
assign TIE_LSIndexed_R = ignore_TIE_LSIndexed_R ? 0 : LSIndexed_C0;
assign TIE_LSOffset_R = ignore_TIE_LSOffset_R ? 0 : VAddrOffset_C0;
assign TIE_MemStoreData128_E = ignore_TIE_MemStoreData128_E ? 0 :
MemDataOut128_C1;
assign TIE_MemStoreData64_E = ignore_TIE_MemStoreData64_E ? 0 :
MemDataOut64_C1;
assign TIE_MemStoreData32_E = ignore_TIE_MemStoreData32_E ? 0 :
MemDataOut32_C1;
assign TIE_MemStoreData16_E = ignore_TIE_MemStoreData16_E ? 0 :
MemDataOut16_C1;
assign TIE_MemStoreData8_E = ignore_TIE_MemStoreData8_E ? 0 : MemDataOut8_C1;
assign MemDataIn128_C2 = TIE_MemLoadData_M;
assign MemDataIn64_C2 = TIE_MemLoadData_M;
assign MemDataIn32_C2 = TIE_MemLoadData_M;
assign MemDataIn16_C2 = TIE_MemLoadData_M;
assign MemDataIn8_C2 = TIE_MemLoadData_M;
assign VAddrIn_C1 = MemOpAddr_E;

// CPEnable and control signals to/from core
assign CPEnable_C1 = CPEnable;
assign TIE_Exception_E = Exception_C1;
assign TIE_ExcCause_E = ExcCause_C1;
assign KillPipe_W = Except_W | Replay_W;
endmodule
module xtdelay1(xtout, xtin, clk);
parameter size = 1;
output [size-1:0] xtout;
input [size-1:0] xtin;
input clk;
    assign xtout = xtin;
endmodule

module xtdelay2(xtout, xtin, clk);
parameter size = 1;
output [size-1:0] xtout;
input [size-1:0] xtin;
input clk;
    assign xtout = xtin;
endmodule

module xtRFenlatch(xtRFenlatchout,xtin,xten,clk);
   parameter size = 32;
   output [size-1:0] xtRFenlatchout;
```

```
    input [size-1:0]  xtin;
    input          xten;
    input          clk;

    reg [size-1:0]    xtRFenlatchout;

  always @(clk or xten or xtin or xtRFenlatchout) begin
   if (clk) begin
      xtRFenlatchout <= #1 (xten) ? xtin : xtRFenlatchout;
   end
  end

endmodule
module xtRFlatch(xtRFlatchout,xtin,clk);
    parameter size = 32;
    output [size-1:0] xtRFlatchout;
    input [size-1:0]  xtin;
    input          clk;

    reg [size-1:0]    xtRFlatchout;

  always @(clk or xtin) begin
   if (clk) begin
    xtRFlatchout <= #1 xtin;
   end
  end

endmodule
module xtadd(xtout, a, b);
    parameter size = 32;

    output [size-1:0] xtout;
    input [size-1:0]  a;
    input [size-1:0]  b;

    assign xtout = a + b;

endmodule
module xtaddc(sum, carry, a, b, c);
    parameter size = 32;

    output [size-1:0] sum;
    output          carry;
    input [size-1:0]  a;
    input [size-1:0]  b;
    input           c;

    wire           junk;

    assign {carry, sum, junk} = {a,c} + {b,c};

endmodule
module xtaddcin(xtout, a, b, c);
    parameter size = 32;

    output [size-1:0] xtout;
    input [size-1:0]  a;
```

```
   input [size-1:0]  b;
   input          c;

   assign xtout = ({a,c} + {b,c}) >> 1;

endmodule
module xtaddcout(sum, carry, a, b);
   parameter size = 1;

   output [size-1:0] sum;
   output            carry;
   input [size-1:0]  a;
   input [size-1:0]  b;

   assign {carry, sum} = a + b;

endmodule
module xtbooth(out, cin, a, b, sign, negate);
parameter size = 16;
output [size+1:0] out;
output cin;
input [size-1:0] a;
input [2:0] b;
input sign, negate;
    wire ase = sign & a[size-1];
    wire [size+1:0] ax1 = {ase, ase, a};
    wire [size+1:0] ax2 = {ase, a, 1'd0};
    wire one = b[1] ^ b[0];
    wire two = b[2] ? ~b[1] & ~b[0] : b[1] & b[0];
    wire cin = negate ? (~b[2] & (b[1] | b[0])) : (b[2] & ~(b[1] & b[0]));
    assign out = {size+2{cin}} ^ (ax1&{size+2{one}} | ax2&{size+2{two}});
endmodule
module xtclock_gate_nor(xtout,xtin1,xtin2);
 output xtout;
 input xtin1,xtin2;

 assign xtout = ~(xtin1 || xtin2);

endmodule
module xtclock_gate_or(xtout,xtin1,xtin2);
 output xtout;
 input xtin1,xtin2;

 assign xtout = (xtin1 || xtin2);

endmodule
module xtcsa (sum, carry, a, b, c);
   parameter size = 1;

   output [size-1:0]    sum;
   output [size-1:0]    carry;
   input [size-1:0]     a;
   input [size-1:0]     b;
   input [size-1:0]     c;

   assign sum = a ^ b ^ c;
   assign carry = (a & b) | (b & c) | (c & a) ;
```

```
endmodule
module xtenflop(xtout, xtin, en, clk);
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  xtin;
   input             en;
   input             clk;
   reg [size-1:0]    tmp;

   assign xtout = tmp;
   always @(posedge clk) begin
      if (en)
      tmp <= #1 xtin;
   end

endmodule
module xtfa(sum, carry, a, b, c);
output sum, carry;
input a, b, c;
    assign sum = a ^ b ^ c;
    assign carry = a & b | a & c | b & c;
endmodule
module xtflop(xtout, xtin, clk);
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  xtin;
   input             clk;
   reg [size-1:0]    tmp;

   assign xtout = tmp;
   always @(posedge clk) begin
      tmp <= #1 xtin;
   end

endmodule
module xtha(sum, carry, a, b);
output sum, carry;
input a, b;
    assign sum = a ^ b;
    assign carry = a & b;
endmodule
module xtinc(xtout, a);
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  a;

   assign xtout = a + 1;

endmodule
module xtmux2e(xtout, a, b, sel);
   parameter size = 32;

   output [size-1:0] xtout;
```

```
    input [size-1:0]  a;
    input [size-1:0]  b;
    input          sel;

    assign xtout = (~sel) ? a : b;

endmodule
module xtmux3e(xtout, a, b, c, sel);
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  a;
   input [size-1:0]  b;
   input [size-1:0]  c;
   input [1:0]         sel;
   reg [size-1:0]    xtout;


   always @(a or b or c or sel) begin
      xtout = sel[1] ? c : (sel[0] ? b : a);
   end
endmodule
module xtmux4e(xtout, a, b, c, d, sel);
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  a;
   input [size-1:0]  b;
   input [size-1:0]  c;
   input [size-1:0]  d;
   input [1:0]         sel;
   reg [size-1:0]    xtout;

   // synopsys infer_mux "xtmux4e"
   always @(sel or a or b or c or d) begin : xtmux4e
      case (sel)          // synopsys parallel_case full_case
      2'b00:
        xtout = a;
      2'b01:
        xtout = b;
      2'b10:
        xtout = c;
      2'b11:
        xtout = d;
      default:
        xtout = {size{1'bx}};
      endcase // case(sel)
   end // always @ (sel or a or b or c or d)

endmodule
module xtnflop(xtout, xtin, clk);
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  xtin;
   input          clk;
   reg [size-1:0]    tmp;
```

```
   assign xtout = tmp;
   always @(negedge clk) begin
      tmp <= #1 xtin;
   end // always @ (negedge clk)

endmodule
module xtscflop(xtout, xtin, clrb, clk);  // sync clear ff
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  xtin;
   input             clrb;
   input             clk;
   reg [size-1:0]    tmp;

   assign xtout = tmp;
   always @(posedge clk) begin
      if (!clrb) tmp <=  0;
      else tmp <= #1 xtin;
   end

endmodule
module xtscenflop(xtout, xtin, en, clrb, clk);  // sync clear
   parameter size = 32;

   output [size-1:0] xtout;
   input [size-1:0]  xtin;
   input             en;
   input                 clrb;
   input             clk;
   reg [size-1:0]    tmp;

   assign xtout = tmp;
   always @(posedge clk) begin
      if (!clrb) tmp <= 0;
      else if (en)
      tmp <= #1 xtin;
   end

endmodule
```

xtensa-gf.h

```
#ifndef XTENSA_NO_INTRINSICS
#ifdef __XTENSA__
/* Do not modify. This is automatically generated.*/
typedef int gf8 __attribute__ ((user("gf8")));

#define GFADD8_ASM(gr, gs, gt)      { \
  __asm__ ("gfadd8      %0,%1,%2" : "=v" (gr) : "v" (gs), "v" (gt)); \
}

#define GFADD8(gs, gt)      ({ \
  gf8 _gr; \
  gf8 _gs = gs; \
```

```
  gf8 _gt = gt; \
  GFADD8_ASM(_gr, _gs, _gt);\
  _gr; \
})

#define GFADD8I_ASM(gr, gs, imm4)      { \
  __asm__ ("gfadd8i     %0,%1,%2" : "=v" (gr) : "v" (gs), "i" (imm4)); \
}

#define GFADD8I(gs, imm4)       ({ \
  gf8 _gr; \
  gf8 _gs = gs; \
  GFADD8I_ASM(_gr, _gs, imm4);\
  _gr; \
})

#define GFMULX8_ASM(gr, gs)      { \
  register int _xt_state asm ("state"); \
  __asm__ ("gfmulx8     %1,%2" : "+t" (_xt_state) , "=v" (gr) : "v" (gs)); \
}

#define GFMULX8(gs)       ({ \
  gf8 _gr; \
  gf8 _gs = gs; \
  GFMULX8_ASM(_gr, _gs);\
  _gr; \
})

#define GFRWMOD8_ASM(gt)      { \
  register int _xt_state asm ("state"); \
  __asm__ ("gfrwmod8    %1" : "+t" (_xt_state) , "=v" (gt) : "1" (gt)); \
}

#define GFRWMOD8(gt)      ({ \
  gf8 _gt = gt; \
  GFRWMOD8_ASM(_gt);\
  gt = _gt; \
})

#define LGF8_I_ASM(gt, ars, imm8)      { \
  __asm__ volatile("lgf8_i    %0,%1,%2" : "=v" (gt) : "a" (ars), "i" (imm8)); \
}

#define LGF8_I(ars, imm8)      ({ \
  gf8 _gt; \
  const unsigned _ars = ars; \
  LGF8_I_ASM(_gt, _ars, imm8);\
  _gt; \
})

#define SGF8_I_ASM(gt, ars, imm8)      { \
  __asm__ volatile("sgf8_i    %0,%1,%2" :  : "v" (gt), "a" (ars), "i" (imm8));
\
}

#define SGF8_I(gt, ars, imm8)      ({ \
  gf8 _gt = gt; \
```

```
  unsigned _ars = ars; \
  SGF8_I_ASM(_gt, _ars, imm8);\
})

#define LGF8_IU_ASM(gt, ars, imm8)     { \
  __asm__ volatile("lgf8_iu   %0,%1,%3" : "=v" (gt), "=a" (ars) : "1" (ars),
"i" (imm8)); \
}

#define LGF8_IU(ars, imm8)     ({ \
  gf8 _gt; \
  unsigned _ars = ars; \
  LGF8_IU_ASM(_gt, _ars, imm8);\
  ars = _ars; \
  _gt; \
})

#define SGF8_IU_ASM(gt, ars, imm8)     { \
  __asm__ volatile("sgf8_iu   %1,%0,%3" : "=a" (ars) : "v" (gt), "0" (ars), "i"
(imm8)); \
}

#define SGF8_IU(gt, ars, imm8)     ({ \
  gf8 _gt = gt; \
  unsigned _ars = ars; \
  SGF8_IU_ASM(_gt, _ars, imm8);\
  ars = _ars; \
})

#define LGF8_X_ASM(gr, ars, art)     { \
  __asm__ volatile("lgf8_x    %0,%1,%2" : "=v" (gr) : "a" (ars), "a" (art)); \
}

#define LGF8_X(ars, art)     ({ \
  gf8 _gr; \
  const unsigned _ars = ars; \
  unsigned _art = art; \
  LGF8_X_ASM(_gr, _ars, _art);\
  _gr; \
})

#define SGF8_X_ASM(gr, ars, art)     { \
  __asm__ volatile("sgf8_x    %0,%1,%2" :  : "v" (gr), "a" (ars), "a" (art)); \
}

#define SGF8_X(gr, ars, art)     ({ \
  gf8 _gr = gr; \
  unsigned _ars = ars; \
  unsigned _art = art; \
  SGF8_X_ASM(_gr, _ars, _art);\
})

#define LGF8_XU_ASM(gr, ars, art)     { \
  __asm__ volatile("lgf8_xu   %0,%1,%3" : "=v" (gr), "=a" (ars) : "1" (ars),
"a" (art)); \
}
```

```
#define LGF8_XU(ars, art)     ({ \
  gf8 _gr; \
  unsigned _ars = ars; \
  unsigned _art = art; \
  LGF8_XU_ASM(_gr, _ars, _art);\
  ars = _ars; \
  _gr; \
})

#define SGF8_XU_ASM(gr, ars, art)     { \
  __asm__ volatile("sgf8_xu   %1,%0,%3" : "=a" (ars) : "v" (gr), "0" (ars), "a"
(art)); \
}

#define SGF8_XU(gr, ars, art)     ({ \
  gf8 _gr = gr; \
  unsigned _ars = ars; \
  unsigned _art = art; \
  SGF8_XU_ASM(_gr, _ars, _art);\
  ars = _ars; \
})

#define RUR0_ASM(arr)     { \
  register int _xt_state asm ("state"); \
  __asm__ ("rur0  %1" : "+t" (_xt_state) , "=a" (arr) : ); \
}

#define RUR0()     ({ \
  unsigned _arr; \
  RUR0_ASM(_arr);\
  _arr; \
})

#define WUR0_ASM(art)     { \
  register int _xt_state asm ("state"); \
  __asm__ ("wur0  %1" : "+t" (_xt_state)  : "a" (art)); \
}

#define WUR0(art)     ({ \
  unsigned _art = art; \
  WUR0_ASM(_art);\
})

#define gf8_loadi(_s, o) ({ \
    gf8 t; \
    gf8 *s = _s; \
    LGF8_I_ASM(t, s, o); \
    t; \
})

#define gf8_storei(_t, _s, o) ({ \
    gf8 t = _t; \
    gf8 *s = _s; \
    SGF8_I_ASM(t, s, o); \
})

#define gf8_move(_r, _s) ({ \
```

```
    gf8 r = _r; \
    gf8 s = _s; \
    GFADD8_ASM(r, s, 0); \
})

#define RUR(n)  ({ \
  int v; \
  register int _xt_state asm ("state"); \
  __asm__ ("rur   %1, %2" : "+t" (_xt_state), "=a" (v) : "i" (n)); \
  v; \
})

#define WUR(v, n) ({ \
  register int _xt_state asm ("state"); \
  __asm__ ("wur   %1, %2" : "+t" (_xt_state) : "a" (v), "i" (n)); \
})
#endif
#endif
```

APPENDIX B

```
#! /usr/xtensa/tools/bin/perl -w

use Getopt::Long;
use strict;

$main::inline_mux_count = 0;
sub inline_mux {
    my($data, $sel, $width, $out, $style, $code) = @_;
    my($i, $n, $n1, $module, $inst, $d, $fail, @data,
@data_uniq);

    $n = @$data;
    if ($style eq "encoded") {
      $module = "xtmux${n}e";
      $fail = 0;
    } elsif ($style eq "priority") {
      $fail = scalar(@$data) != scalar(@$sel)+1;
      $module = "xtmux${n}p";
    } elsif ($style eq "selector") {
      $fail = scalar(@$data) != scalar(@$sel);
      $module = "xtmux${n}";
    } else {
      die "inline_mux: bad style $style";
    }

    if ($fail) {
      die "inline_mux: data / selection mismatch for $style $n";
    }

    if ($n == 0) {
      print "    assign $out = 0;\n";
    } elsif ($n == 1) {
      print "    assign $out = " . (shift @$data) . ";\n";
    } else {
      @data_uniq = uniq(@$data);
      $n1 = @data_uniq;
      if ($style eq "priority" && ($n1 != $n || defined $code)) {
          if (! defined $code) {
            for($i = 0; $i < $n1; $i++) {
                $code->{$data_uniq[$i]} = $i;
            }
          }
          @data = sort { $code->{$a} <=> $code->{$b} }
@data_uniq;
          print "    wire [" . (ceil_log2($n1)-1) . ":0]
${out}_sel =\n";
          for($i = 0; $i < $n-1; $i++) {
            print "        $sel->[$i] ? $code->{$data->[$i]}
:\n";
          }
          print "        $code->{$data->[$n-1]};\n";
          inline_mux(\@data, "${out}_sel", $width, $out,
"encoded");
      } else {
          # drop an instance of the mux
          $inst = $main::inline_mux_count++;
          print "    $module #($width) m$inst($out";
```

```
            print map(", $_", @$data);
            if ($style eq "priority" || $style eq "selector") {
              print map(", $_", @$sel);
              print ");\n";
            } else {
              print ", $sel);\n";
            }
        }
    }
}


# min of a list
sub min {
    my($min, $v);
    $min = $_[0];
    foreach $v (@_) {
      $min = $v < $min ? $v : $min;
    }
    return $min;
}


# max of a list
sub max {
    my($max, $v);
    $max = $_[0];
    foreach $v (@_) {
      $max = $v > $max ? $v : $max;
    }
    return $max;
}


# ceil(log2(x))
sub ceil_log2 {
    my($x) = @_;
    my($n);
    for($n = 0, $x -= 1; $x > 0; $x >>= 1, $n++) {
    }
    return $n;
}


# 2^x
sub pow2 {
    my($x) = @_;
    return 1 << $x;
}


# uniqify an array
sub uniq {
    my(%seen);
    return grep(! $seen{$_}++, @_);
}
```

```
# difference between two arrays
sub diff {
    my($aref, $bref) = @_;
    my(%hash);
    grep($hash{$_} = 1, @$bref);
    return grep(! defined $hash{$_}, @$aref);
}


sub wfield {
    my($name, $port, $stage) = @_;
    $name = "$port->{NAME}_$name";
    return $stage >= 0 ? "${name}_C$stage" : $name;
}


sub rfield {
    my($name, $port, $stage) = @_;
    $name = "$port->{NAME}_$name";
    return $stage >= 0 ? "${name}_C$stage" : $name;
}


sub write_def {
    my($write_port, $stage) = @_;
    return grep($_ == $stage, @{$write_port->{DEF}});
}


sub read_use {
    my($read_port, $stage) = @_;
    return grep($_ == $stage, @{$read_port->{USE}});
}


sub init_print_break {
    my($indent) = @_;
    $main::col = 0;
    $main::indent = $indent;
}

sub print_break {
    my($d) = @_;
    if ($main::col + length($d) + 1 >= 85) {
      $main::col = 4;
      print ("\n" . (' ' x $main::indent));
    }
    print "$d";
    $main::col += length($d) + 1;
}
```

```
        sub doc {
            my($a) = <<'END_OF_DOCUMENTATION';

        The pipelined register file instantiates a number of pipelined
register file
        banks, each of which contains a register file core.

        The core is a simple multiple-read port multiple-write port
register file.  The
        address size is $rf->{ADDR_SIZE} (lg2 $rf->{MIN_HEIHGT}) and its
declaration is
        $rf->{ADDR_DECL}.  The data size is $rf->{DECL_SIZE} ($rf-
>{MIN_WIDTH}) and its
        declaration is $rf->{DECL_DECL}.

        Multiple banks are used to support multiple widths for read and
write ports.
        We build NUM_BANK ($rf->MAX_WIDTH / $rf->MIN_WIDTH) pipelined
register banks,
        each of which has MIN_HEIGHT words and MIN_WIDTH bits in each
word.  Each width
        must be a power of 2 multiple of the minimum width; in
particular, NUM_BANK
        must also be a power of 2.

        A final read alignment mux looks at the low-order address bits
and the
        read-width mask to mux the correct data onto the output.  This
splits the
        address into HI_ADDR_SIZE and LO_ADDR_SIZE fields.  The high
order bits go
        directly to the register file core; the low address bits are fed
to the
        alignment mux.  The read output is always MAX_WIDTH in size and
smaller data
        values are shifted to the LSB of the output word.

        As a concrete example, consider a register file of size 1024 bits
(32x32) with
        read widths of 32 and 128.

            NUM_BANK = 4
            MIN_HEIGHT = 32
            MIN_WIDTH = 32
            MAX_HEIGHT = 8
            MAX_WIDTH = 128
            ADDR_SIZE = 5
            ADDR_DECL = [4:0]
            WORD_SIZE = 32
            WORD_DECL = [31:0]
            HI_ADDR_SIZE = 3
            LO_ADDR_SIZE = 2

        The read mask is:
              11    to read width 32
              10    to read width 64 (not used in this case)
```

```
        00    to read width 128

END_OF_DOCUMENTATION
    return $a;
}



sub derive_constants {
    my($rf) = @_;
    my($read_port, $write_port, $n, $w, @width);

    # determine parameters for register file banks
    foreach $read_port (@{$rf->{READ_PORT}}) {
      push(@width, @{$read_port->{WIDTH}});
    }
    foreach $write_port (@{$rf->{WRITE_PORT}}) {
      push(@width, @{$write_port->{WIDTH}});
    }
    @width = sort {$a <=> $b} (&uniq(@width));

    $rf->{MIN_WIDTH} = $width[0];
    $rf->{MAX_WIDTH} = $width[$#width];
    $rf->{MIN_HEIGHT} = $rf->{SIZE} / $rf->{MAX_WIDTH};
    $rf->{MAX_HEIGHT} = $rf->{SIZE} / $rf->{MIN_WIDTH};
    $rf->{NUM_BANK} = $rf->{MAX_WIDTH} / $rf->{MIN_WIDTH};

    foreach $w (@width) {
      $n = $w / $rf->{MIN_WIDTH};
      if ($n != pow2(ceil_log2($n))) {
          die "width $w not valid multiple of $rf-
>{MIN_WIDTH}\n";
      }
    }


    # register file core parameters
    $rf->{ADDR_SIZE} = ceil_log2($rf->{MIN_HEIGHT});
    $rf->{ADDR_DECL} = $rf->{ADDR_SIZE} > 0 ? "[" . ($rf-
>{ADDR_SIZE}-1) . ":0]" : "";
    $rf->{WORD_SIZE} = $rf->{MIN_WIDTH};
    $rf->{WORD_DECL} = $rf->{WORD_SIZE} > 0 ? "[" . ($rf-
>{WORD_SIZE}-1) . ":0]" : "";

    $rf->{HI_ADDR_SIZE} = ceil_log2($rf->{MAX_HEIGHT});
    $rf->{LO_ADDR_SIZE} = $rf->{HI_ADDR_SIZE} - $rf->{ADDR_SIZE};
    $rf->{FULL_WORD_SIZE} = $rf->{MAX_WIDTH};
    $rf->{FULL_WORD_DECL} = $rf->{FULL_WORD_SIZE} > 0 ? "[" .
($rf->{FULL_WORD_SIZE}-1) . ":0]" : "";


    $rf->{MAX_LATENCY} = 0;
    foreach $write_port (@{$rf->{WRITE_PORT}}) {
      my(@def) = sort(&uniq(@{$write_port->{DEF}}));
      $write_port->{DEF} = \@def;
      $write_port->{MAX_DEF} = &max(2, @{$write_port->{DEF}});
      $write_port->{MAX_WIDTH} = max(@{$write_port->{WIDTH}});
```

```
        $rf->{MAX_LATENCY} = max($rf->{MAX_LATENCY}, $write_port-
>{MAX_DEF});
      }
      foreach $read_port (@{$rf->{READ_PORT}}) {
        my(@use) = sort(&uniq(@{$read_port->{USE}}));
        $read_port->{USE} = \@use;
        $read_port->{MIN_USE} = min(@{$read_port->{USE}});
        $read_port->{MAX_USE} = max(@{$read_port->{USE}});
        $read_port->{MAX_WIDTH} = max(@{$read_port->{WIDTH}});
      }

      $rf->{NUM_TEST_VECTOR} = $rf->{NUM_TEST_VECTOR} || 1000;
      $rf->{USE_LATCHES} = $rf->{USE_LATCHES} || 1;
      $rf->{TEST_TRANSPARENT_LATCHES} = $rf-
>{TEST_TRANSPARENT_LATCHES} || 0;
      if ($rf->{TRANSPARENT_LATCH_MODE}) {   # an old name for it
        $rf->{TEST_TRANSPARENT_LATCHES} = 1;
      }
      $rf->{DESIGN_PREFIX} = $rf->{DESIGN_PREFIX} || "";
  }


  sub write_regfile {
      my($rf) = @_;
      my($lo_addr_decl, @iolist, $s, $i, $j, $h, $l, $w);
      my(@defer, $read_port, $write_port);

      $lo_addr_decl = $rf->{LO_ADDR_SIZE} > 0 ? "[" . ($rf-
>{LO_ADDR_SIZE}-1) . ":0]" : "";

      init_print_break(2);
      print_break("module $rf->{DESIGN_PREFIX}$rf->{NAME}(");
      foreach $read_port (@{$rf->{READ_PORT}}) {
        foreach $s (@{$read_port->{USE}}) {
            my($data) = rfield("data", $read_port, $s);
            my($decl) = "[" . ($read_port->{MAX_WIDTH} - 1) .
":0]";

            print_break("$data, ");
            push(@iolist, "    output $decl $data;\n");
        }
        # don't need an address for a single word register file
        if ($rf->{HI_ADDR_SIZE} > 0) {
            my($addr) = rfield("addr", $read_port, 0);
            my($decl) = "[" . ($rf->{HI_ADDR_SIZE} - 1) . ":0]";
            print_break("$addr, ");
            push(@iolist, "    input $decl $addr;\n");
        } else {
            my($addr) = rfield("addr", $read_port, 0);
            push(@defer, "    wire $addr = 0;\n");
        }
        foreach $w (@{$read_port->{WIDTH}}) {
            my($width) = rfield("width$w", $read_port, 0);
            print_break("$width, ");
            push(@iolist, "    input $width;\n");
        }
        foreach $s (@{$read_port->{USE}}) {
```

```
          my($use) = rfield("use$s", $read_port, 0);
          print_break("$use, ");
          push(@iolist, "    input $use;\n");
       }
    }

    foreach $write_port (@{$rf->{WRITE_PORT}}) {
      # don't need an address for a single word register file
      if ($rf->{HI_ADDR_SIZE} > 0) {
          my($addr) = wfield("addr", $write_port, 0);
          my($decl) = "[" . ($rf->{HI_ADDR_SIZE} - 1) . ":0]";
          print_break("$addr, ");
          push(@iolist, "    input $decl $addr;\n");
      } else {
          my($addr) = rfield("addr", $write_port, 0);
          push(@defer, "    wire $addr = 0;\n");
      }
      foreach $w (@{$write_port->{WIDTH}}) {
          my($width) = rfield("width$w", $write_port, 0);
          print_break("$width, ");
          push(@iolist, "    input $width;\n");
      }
      foreach $s (@{$write_port->{DEF}}) {
          my($def) = wfield("def$s", $write_port, 0);
          print_break("$def, ");
          push(@iolist, "    input $def;\n");
      }
      foreach $w (@{$write_port->{WIDTH}}) {
          foreach $s (@{$write_port->{DEF}}) {
            my($data) = wfield("data$w", $write_port, $s);
            my($decl) = "[" . ($w - 1) . ":0]";
            print_break("$data, ");
            push(@iolist, "    input $decl $data;\n");
          }
      }
      foreach $s (1 .. $write_port->{MAX_DEF}) {
          my($wen) = wfield("wen", $write_port, $s);
          if ($s > &max(@{$write_port->{DEF}})) {
            push(@defer, "    wire $wen = 1'd1;\n");
          } else {
            print_break("$wen, ");
            push(@iolist, "    input $wen;\n");
          }
      }
    }

    print_break("Kill_E, ");
    push(@iolist, "    input Kill_E;\n");

    print_break("KillPipe_W, ");
    push(@iolist, "    input KillPipe_W;\n");

    print_break("Stall_R, ");
    push(@iolist, "    output Stall_R;\n");

    if ($rf->{USE_LATCHES} && $rf->{TEST_TRANSPARENT_LATCHES}) {
      print_break("TMode, ");
```

```
      push(@iolist, "input TMode;\n");
    }

    print_break("clk);\n");
    push(@iolist, "    input clk;\n");

    print join('', @iolist);
    print "\n";

    print join('', @defer);
    print "\n";


    foreach $read_port (@{$rf->{READ_PORT}}) {
      print "    /*" . ('*' x 70) . "\n";
      print "           READ PORT $read_port->{NAME}\n";
      print "     *" . ('*' x 70) . "/\n";
      if ($rf->{LO_ADDR_SIZE} > 0) {
          my(@data, @sel);
          foreach $w (@{$read_port->{WIDTH}}) {
            my($width) = rfield("width$w", $read_port, 0);
            my($mask) = ~($w / $rf->{MIN_WIDTH} - 1) & ((1 <<
$rf->{LO_ADDR_SIZE}) - 1);
            push(@data, $rf->{LO_ADDR_SIZE} . "'d" . $mask);
            push(@sel, $width);
          }
          my($addr_mask) = rfield("addr_mask", $read_port, 0);
          print "    wire $lo_addr_decl $addr_mask;\n";
          inline_mux(\@data, \@sel, $rf->{LO_ADDR_SIZE},
$addr_mask, "selector");
      } else {
          my($addr_mask) = rfield("addr_mask", $read_port, 0);
          print "    wire $addr_mask = 0;\n";
      }
      print "\n";

      print "    // masked address pipeline\n";
      if ($rf->{LO_ADDR_SIZE} > 0) {
          my($addr) = rfield("addr", $read_port, 0);
          my($maddr) = rfield("maddr", $read_port, 0);
          my($addr_mask) = rfield("addr_mask", $read_port, 0);
          print "    wire $lo_addr_decl $maddr = $addr &
$addr_mask;\n";
          for($s = 1; $s <= $read_port->{MAX_USE}; $s++) {
            my($maddr) = rfield("maddr", $read_port, $s);
            print "    wire $lo_addr_decl $maddr;\n";
          }
          for($s = 1; $s <= $read_port->{MAX_USE}; $s++) {
            my($maddr) = rfield("maddr", $read_port, $s-1);
            my($maddr1) = rfield("maddr", $read_port, $s);
            print "    xtdelay1 #($rf->{LO_ADDR_SIZE})
i$maddr1($maddr1, $maddr, clk);\n";
          }
      } else {
          my($maddr) = rfield("maddr", $read_port, 0);
          print "    wire $maddr = 0;\n";
      }
```

```
            print "\n";

            print "    // bank-qualified use\n";
            foreach $s (@{$read_port->{USE}}) {
                foreach $i (0 .. $rf->{NUM_BANK}-1) {
                  my($use) = rfield("use$s", $read_port, 0);
                  my($maddr) = rfield("maddr", $read_port, 0);
                  my($addr_mask) = rfield("addr_mask", $read_port, 0);
                  my($use_banki) = rfield("use$s" . "_bank$i",
$read_port, 0);
                  print "    wire $use_banki = ($use & ($maddr == ($i &
$addr_mask));\n";
                }
            }
            print "\n";

            # determine which banks need to be muxed into which output
ports
            my(@align);
            for($i = 0; $i < $rf->{NUM_BANK}; $i++) {
                $align[$i] = [ ];
            }
            for($w = 1; $w <= $rf->{NUM_BANK}; $w *= 2) {
                # does this port need this read-width?
                if (grep($_ == $w * $rf->{MIN_WIDTH}, @{$read_port-
>{WIDTH}})) {
                  for($j = 0; $j < $rf->{NUM_BANK}; $j += $w) {
                      for($i = 0; $i < $w; $i++) {
                        push(@{$align[$i]}, $i+$j);
                      }
                  }
                }
            }

            for($i = 0; $i < $rf->{NUM_BANK}; $i++) {
                @{$align[$i]} = sort {$a <=> $b}
(&uniq(@{$align[$i]}));
            }

#           print STDOUT "Read table\n";
#           for($i = 0; $i < $rf->{NUM_BANK}; $i++) {
#               print STDOUT "set $i: " . join(' ', @{$align[$i]}) .
"\n";
#           }

            foreach $s (@{$read_port->{USE}}) {
                print "    // alignment mux for use $s\n";
                for($i = 0; $i < $rf->{NUM_BANK}; $i++) {
                  my($data_banki) = rfield("data_bank$i", $read_port,
$s);

                  print "    wire $rf->{WORD_DECL} $data_banki;\n";
                }
                for($i = 0; $i < $rf->{NUM_BANK}; $i++) {
                  my(@data);
                  foreach $j (@{$align[$i]}) {
                      my($data_bankj) = rfield("data_bank$j",
$read_port, $s);
```

```
                    push(@data, $data_bankj);
                }

                $h = $rf->{LO_ADDR_SIZE} - 1;
                $l = $rf->{LO_ADDR_SIZE} - ceil_log2($#data + 1);
                my($sel) = rfield("maddr", $read_port, $s) .
"[$h:$l]";

                $h = $rf->{MIN_WIDTH} * ($i+1) - 1;
                $l = $rf->{MIN_WIDTH} * $i;
                my($data) = rfield("data", $read_port, $s) .
"[$h:$l]";
                my($prefix) = rfield("align$i", $read_port, $s);
                if (@data > 0) {
                    inline_mux(\@data, $sel, $rf->{WORD_SIZE}, $data,
"encoded");
                }
              }
              print "\n";
          }
          print "\n";
        }


        foreach $write_port (@{$rf->{WRITE_PORT}}) {
          print "    /*" . ('*' x 70) . "\n";
          print "            WRITE PORT $write_port->{NAME}\n";
          print "     *" . ('*' x 70) . "/\n";
          if ($rf->{LO_ADDR_SIZE} > 0) {
              my(@data, @sel);
              foreach $w (@{$write_port->{WIDTH}}) {
                my($width) = wfield("width$w", $write_port, 0);
                my($mask) = ~($w / $rf->{MIN_WIDTH} - 1) & ((1 <<
$rf->{LO_ADDR_SIZE}) - 1);
                push(@data, $rf->{LO_ADDR_SIZE} . "'d" . $mask);
                push(@sel, $width);
              }
              my($addr_mask) = wfield("addr_mask", $write_port, 0);
              print "    wire $lo_addr_decl $addr_mask;\n";
              inline_mux(\@data, \@sel, $rf->{LO_ADDR_SIZE},
$addr_mask, "selector");
          } else {
              my($addr_mask) = wfield("addr_mask", $write_port, 0);
              print "    wire $addr_mask = 0;\n";
          }
          print "\n";

          if (@{$write_port->{WIDTH}} > 1) {
              print "    // width pipeline\n";
              foreach $w (@{$write_port->{WIDTH}}) {
                for($s = 1; $s <= $write_port->{MAX_DEF}; $s++) {
                    my($width) = wfield("width$w", $write_port, $s);
                    print "    wire $width;\n";
                }
                for($s = 1; $s <= $write_port->{MAX_DEF}; $s++) {
                    my($width) = wfield("width$w", $write_port, $s-
1);
```

```
                    my($width1) = wfield("width$w", $write_port, $s);
                    print "    xtdelay1 #(1) i$width1($width1,
$width, clk);\n";
                  }
                }
                print "\n";
              }

              print "    // bank-qualified write def for port
$write_port->{NAME}\n";
              foreach $s (@{$write_port->{DEF}}) {
                foreach $i (0 .. $rf->{NUM_BANK}-1) {
                  my($def) = wfield("def$s", $write_port, 0);
                  my($addr) = wfield("addr", $write_port, 0);
                  my($addr_mask) = wfield("addr_mask", $write_port, 0);
                  my($def_banki) = wfield("def$s" . "_bank$i",
$write_port, 0);
                  print "    wire $def_banki = ($def & (($addr &
$addr_mask) == ($i & $addr_mask)));\n";
                }
              }
              print "\n";

              foreach $s (@{$write_port->{DEF}}) {
                my(@data, @sel);
                print "    // write mux for def $s\n";
                my($wdata) = wfield("wdata", $write_port, $s);
                foreach $w (@{$write_port->{WIDTH}}) {
                  $i = $rf->{MAX_WIDTH} / $w;
                  my($width) = wfield("width$w", $write_port, $s);
                  my($data) = wfield("data$w", $write_port, $s);
                  $data = "{" . $i . "{$data" . "[" . ($w-1) . ":0]}}";
                  push(@data, $data);
                  push(@sel, $width);
                }
                print "    wire $rf->{FULL_WORD_DECL} $wdata;\n";
                inline_mux(\@data, \@sel, $rf->{FULL_WORD_SIZE},
$wdata, "selector");
                print "\n";
              }
              print "\n";
            }


            # drop n copies of the pipelined regfile
            print "    /*" . ('*' x 70) . "\n";
            print "         PIPELINED BANK\n";
            print "     *" . ('*' x 70) . "/\n";
            for($i = 0; $i < $rf->{NUM_BANK}; $i++) {
              init_print_break(8);
              print_break("    $rf->{DESIGN_PREFIX}$rf->{NAME}_bank $rf-
>{NAME}_bank$i(");
              foreach $read_port (@{$rf->{READ_PORT}}) {
                foreach $s (@{$read_port->{USE}}) {
                  my($data_banki) = rfield("data_bank$i", $read_port,
$s);

                  print_break("$data_banki, ");
```

```
        }
        # don't need an address for a single word register file
        if ($rf->{ADDR_SIZE} > 0) {
          my($addr) = rfield("addr", $read_port, 0);
          my($decl) = "[" . ($rf->{HI_ADDR_SIZE}-1) . ":$rf-
>{LO_ADDR_SIZE}]";
          print_break("$addr$decl, ");
        }
        foreach $s (@{$read_port->{USE}}) {
          my($use_banki) = rfield("use$s" . "_bank$i",
$read_port, 0);
          print_break("$use_banki, ");
        }
      }

      foreach $write_port (@{$rf->{WRITE_PORT}}) {
        if ($rf->{ADDR_SIZE} > 0) {
          my($addr) = wfield("addr", $write_port, 0);
          my($decl) = "[" . ($rf->{HI_ADDR_SIZE}-1) . ":$rf-
>{LO_ADDR_SIZE}]";
          print_break("$addr$decl, ");
        }
        foreach $s (@{$write_port->{DEF}}) {
          my($def_banki) = wfield("def$s" . "_bank$i",
$write_port, 0);
          print_break("$def_banki, ");
        }
        foreach $s (@{$write_port->{DEF}}) {
          my($wdata) = wfield("wdata", $write_port, $s);
          $h = $rf->{MIN_WIDTH} * ($i+1) - 1;
          $l = $rf->{MIN_WIDTH} * $i;
          print_break("$wdata" . "[$h:$l], ");
        }
        foreach $s (1 .. $write_port->{MAX_DEF}) {
          my($wen) = wfield("wen", $write_port, $s);
          print_break("$wen, ");
        }
      }

      print_break("Kill_E, ");
      print_break("KillPipe_W, ");
      print_break("Stall_R$i, ");
      if ($rf->{USE_LATCHES} && $rf->{TEST_TRANSPARENT_LATCHES})
{
        print_break("TMode, ");
      }
      print_break("clk);\n");
      print "\n";
    }

    print "    assign Stall_R =";
    for($i = 0; $i < $rf->{NUM_BANK}; $i++) {
      print " Stall_R$i |";
    }
    print " 1'b0;\n";
    print "\n";
```

```
    print "endmodule\n";
}



sub write_regfile_bank {
    my($rf) = @_;
    my(@defer, @iolist, $s, $sl, $rs, $ws, $i, $j, $read_port,
$write_port, $result);

    init_print_break(2);
    print_break("module $rf->{DESIGN_PREFIX}$rf->{NAME}_bank(");

    # read port I/O list
    foreach $read_port (@{$rf->{READ_PORT}}) {
      foreach $s (@{$read_port->{USE}}) {
          my($data) = rfield("data", $read_port, $s);
          print_break("$data, ");
          push(@iolist, "    output $rf->{WORD_DECL} $data;\n");
      }

      # don't need an address for a single word register file
      my($addr) = rfield("addr", $read_port, 0);
      if ($rf->{ADDR_SIZE} > 0) {
          print_break("$addr, ");
          push(@iolist, "    input $rf->{ADDR_DECL} $addr;\n");
      } else {
          push(@defer, "    wire $rf->{ADDR_DECL} $addr = 0;\n");
      }

      foreach $s (1 .. $rf->{MAX_LATENCY}) {
          my($use) = rfield("use$s", $read_port, 0);
          if (read_use($read_port, $s)) {
            print_break("$use, ");
            push(@iolist, "    input $use;\n");
          } else {
            push(@defer, "    wire $use = 0;\n");
          }
      }
    }


    # write port I/O list
    foreach $write_port (@{$rf->{WRITE_PORT}}) {
      my($addr) = wfield("addr", $write_port, 0);
      if ($rf->{ADDR_SIZE} > 0) {
          print_break("$addr, ");
          push(@iolist, "    input $rf->{ADDR_DECL} $addr;\n");
      } else {
          push(@defer, "    wire $rf->{ADDR_DECL} $addr = 0;\n");
      }

      foreach $s (1 .. $write_port->{MAX_DEF}) {
          my($def) = wfield("def$s", $write_port, 0);
          if (write_def($write_port, $s)) {
            print_break("$def, ");
```

```
          push(@iolist, "    input $def;\n");
        } else {
          push(@defer, "    wire $def = 0;\n");
        }
    }

    foreach $s (1 .. $write_port->{MAX_DEF}) {
        my($data) = wfield("data", $write_port, $s);
        if (write_def($write_port, $s)) {
          print_break("$data, ");
          push(@iolist, "    input $rf->{WORD_DECL} $data;\n");
        } else {
          push(@defer, "    wire $rf->{WORD_DECL} $data =
0;\n");
        }
    }

    foreach $s (1 .. $write_port->{MAX_DEF}) {
        my($wen) = wfield("wen", $write_port, $s);
        print_break("$wen, ");
        push(@iolist, "    input $wen;\n");
    }
  }

  print_break("Kill_E, ");
  push(@iolist, "    input Kill_E;\n");

  print_break("KillPipe_W, ");
  push(@iolist, "    input KillPipe_W;\n");

  print_break("Stall_R, ");
  push(@iolist, "    output Stall_R;\n");

  if ($rf->{USE_LATCHES} && $rf->{TEST_TRANSPARENT_LATCHES}) {
    print_break("TMode, ");
    push(@iolist, "input TMode;\n");
  }

  print_break("clk);\n");
  push(@iolist, "    input clk;\n");

  print join('', @iolist);
  print "\n";

  print join('', @defer);
  print "\n";

  for($s = 0; $s <= $rf->{MAX_LATENCY}+1; $s++) {
    # can't kill after commit point which is C3
    my($kill) = "kill_C$s";
    my($value) =
        $s == 1 ? "KillPipe_W | Kill_E" :
        $s <= 3 ? "KillPipe_W" :
        "1'b0";
    print "    wire $kill = $value;\n";
  }
  print "\n";
```

```
####################################################################
####
          #  Write-port information

####################################################################
####
          foreach $write_port (@{$rf->{WRITE_PORT}}) {
            # write definition pipeline
            print "    // write definition pipeline\n";
            for($s = 1; $s <= $write_port->{MAX_DEF}; $s++) {
                for($i = $s; $i <= $write_port->{MAX_DEF}; $i++) {
                  my($wen) = $s == 1 ? "1" : wfield("wen", $write_port,
 $s-1);

                  my($def) = wfield("def$i", $write_port, $s-1);
                  my($ns_def) = wfield("ns_def$i", $write_port, $s-1);
                  my($def1) = wfield("def$i", $write_port, $s);
                  my($kill) = "kill_C" . ($s-1);
                  if (write_def($write_port, $i)) {
                      print "    wire $ns_def = $def & $wen &
 ~$kill;\n";

                      print "    xtdelay1 #(1) i$def1($def1, $ns_def,
 clk);\n";

                  } else {
                      print "    wire $ns_def = 0;\n";
                      print "    wire $def1 = 0;\n";
                  }
                }
          }
          print "\n";

          # write enable pipeline
          print "    // write enable pipeline\n";
          for($s = 1; $s <= $write_port->{MAX_DEF}; $s++) {
              my $we1 = wfield("we", $write_port, $s+1);
              print "    wire $we1;\n";
          }
          for($s = 1; $s <= $write_port->{MAX_DEF}+1; $s++) {
              my $first = $s == 1;
              my $last = $s == $write_port->{MAX_DEF} + 1;
              my $we = $first ? "1'd0" : wfield("we", $write_port,
 $s);

              my $def = $last ? "1'd0" : wfield("def$s", $write_port,
 $s);

              my $wen = $last ? "1'd0" : wfield("wen", $write_port,
 $s);

              my $kill = "kill_C$s";
              my $ns_we = wfield("ns_we", $write_port, $s);
              print "    wire $ns_we = ($we | ($def & $wen)) &
 ~$kill;\n";
          }
          for($s = 1; $s <= $write_port->{MAX_DEF}; $s++) {
              my $ns_we = wfield("ns_we", $write_port, $s);
              my $we1 = wfield("we", $write_port, $s+1);
              print "    xtdelay1 #(1) i$we1($we1, $ns_we, clk);\n";
```

```
            }
            print "\n";

            # Write address pipeline
            print "    // write address pipeline\n";
            for($s = 1; $s <= $write_port->{MAX_DEF}+1; $s++) {
                my $addr = wfield("addr", $write_port, $s);
                print "    wire $rf->{ADDR_DECL} $addr;\n";
            }
            for($s = 1; $s <= $write_port->{MAX_DEF}+1; $s++) {
                my $addr = wfield("addr", $write_port, $s-1);
                my $addr1 = wfield("addr", $write_port, $s);
                if ($rf->{ADDR_SIZE} == 0) {
                  print "    assign $addr1 = 0;\n";
                } else {
                  print "    xtdelay1 #($rf->{ADDR_SIZE})
i$addr1($addr1, $addr, clk);\n";
                }
            }
            print "\n";


            # Write data pipeline
            print "    // write data pipeline\n";
            for($s = 1; $s <= $write_port->{MAX_DEF}; $s++) {
                my $result1 = wfield("result", $write_port, $s+1);
                print "    wire $rf->{WORD_DECL} $result1;\n";
            }
            for($s = 1; $s <= $write_port->{MAX_DEF}+1; $s++) {
                my $result = wfield("result", $write_port, $s);
                my $data = wfield("data", $write_port, $s);
                my $sel = wfield("def$s", $write_port, $s);
                my $mux = wfield("mux", $write_port, $s);
                if ($s == 1) {
                  print "    wire $rf->{WORD_DECL} $mux = $data;\n";
                } elsif ($s == $write_port->{MAX_DEF}+1) {
                  print "    wire $rf->{WORD_DECL} $mux = $result;\n";
                } else {
                  print "    wire $rf->{WORD_DECL} $mux = $sel ? $data
: $result;\n";
    #             print "    xtmux2e #($rf->{WORD_SIZE}) i$mux($mux,
$result, $data, $sel);\n";
                }
            }
            for($s = 1; $s <= $write_port->{MAX_DEF}; $s++) {
                my $mux = wfield("mux", $write_port, $s);
                my $result1 = wfield("result", $write_port, $s+1);
                print "    xtdelay1 #($rf->{WORD_SIZE})
i$result1($result1, $mux, clk);\n";
            }
            print "\n";
        }


#######################################################################
####
```

```
        #  Read-port information
##################################################################################
####
        foreach $read_port (@{$rf->{READ_PORT}}) {
          # need to declare read data which aren't ports
          for($s = $read_port->{MIN_USE} - 1; $s <= $read_port-
>{MAX_USE}; $s++) {
              if (! read_use($read_port, $s)) {
                my($data) = rfield("data", $read_port, $s);
                print "    wire $rf->{WORD_DECL} $data;\n";
              }
          }
        }
        print "\n";

        foreach $read_port (@{$rf->{READ_PORT}}) {
          if ($read_port->{MAX_USE} >= 2) {
              print "    // read address pipeline for port
$read_port->{NAME}\n";
              for($s = 1; $s <= $read_port->{MAX_USE}-1; $s++) {
                my $addr1 = rfield("addr", $read_port, $s);
                print "    wire $rf->{ADDR_DECL} $addr1;\n";
              }
              for($s = 1; $s <= $read_port->{MAX_USE}-1; $s++) {
                my $addr = rfield("addr", $read_port, $s-1);
                my $addr1 = rfield("addr", $read_port, $s);
                if ($rf->{ADDR_SIZE} == 0) {
                    print "    assign $addr1 = 0;\n";
                } else {
                    print "    xtdelay1 #($rf->{ADDR_SIZE})
i$addr1($addr1, $addr, clk);\n";
                }
              }
              print "\n";
        }

    $rs = <<DOCUMENTATION;
        Bypass logic generation is somewhat tricky.  For the first
use
        (typically use1) the data comes from

            (a) write data coming from the datapath (wr0_data_Ci,
i=1..n)
            (b) data stored in the write pipeline (wr0_result_Cn,
i=2..n+1)
            (c) the register file (rd0_data_C0)

        For later uses (e.g., use 2) the data comes from

            (a) write data coming from the datapath (wr0_data_Ci,
i=2..n)
            (b) the read pipeline previous stage (rd0_data_C{i-1})

        To avoid WAW hazards, there is a defined priority on this
.data.
```

Consider a use 1,2,3,4 read pipe and a def 1,2,3,4 write pipe.

The priority order for use 1 is:

wr0_data_C1,
wr0_data_C2,
wr0_result_C2,
wr0_data_C3,
wr0_result_C3,
wr0_data_C4,
wr0_result_C4,
wr0_result_C5,
register file.

The priority order for use 2 is similar, except for all places where the write pipeline would be used, we use the previous stage read pipeline instead. This is because the data stored in the write pipeline has already been bypassed into the read pipeline earlier. Hence, the unique sources are wr0_data_C2, wr0_data_C3, wr0_data_C4, rd0_data_C1 with a priority order of:

wr0_data_C1,
wr0_data_C2,
rd0_data_C1,
wr0_data_C3,
rd0_data_C1,
wr0_data_C4,
rd0_data_C1,
rd0_data_C1,
rd0_data_C1.

Because of all of the write pipeline data is available very early, we build a special mux for the first stage bypass. We first mux together all of the stored data in the write pipe with the read data from the register file. Then we mux together all of the data coming from the datapath. Finally, we select between these two.

DOCUMENTATION

```
        if ($main::verify) {
            for($rs = $read_port->{MIN_USE}-1; $rs <= $read_port->{MAX_USE}-1; $rs++) {
                my $rdata = rfield("data", $read_port, $rs);
                my $rdata1 = rfield("data", $read_port, $rs+1);
                print "    xtdelay1 #($rf->{WORD_SIZE}) i$rdata1($rdata1, $rdata, clk);\n";
            }
            print "\n";
```

```
              } else {
                  print "    // Read bypass controls for port $read_port-
>{NAME}\n";

                  # bypass the data being defined in stage $ws
                  for($rs = $read_port->{MIN_USE}-1; $rs <= $read_port-
>{MAX_USE}-1; $rs++) {
                      for($ws = $rs+1; $ws <= $rf->{MAX_LATENCY}+1; $ws++)
{
                          foreach $write_port (@{$rf->{WRITE_PORT}}) {
                            if (write_def($write_port, $ws)) {
                                my $waddr = wfield("addr", $write_port,
$ws);
                                my $raddr = rfield("addr", $read_port,
$rs);
                                my $def = wfield("def$ws", $write_port,
$ws);
                                my $wen = wfield("wen", $write_port, $ws);
                                my $kill = "kill_C$ws";
                                my $bypass = "bypass_data_$read_port-
>{NAME}_C$rs\_$write_port->{NAME}_C$ws";
                                print "    wire $bypass = ($waddr ==
$raddr) & $def & $wen & ~$kill;\n";
                            }
                          }
                      }
                  }

                  # bypass the old data in the write pipeline in stage
$ws
                  for($rs = $read_port->{MIN_USE}-1; $rs <= $read_port-
>{MAX_USE}-1; $rs++) {
                      for($ws = $rs+1; $ws <= $rf->{MAX_LATENCY}+1; $ws++)
{
                          foreach $write_port (@{$rf->{WRITE_PORT}}) {
                            if ($ws > 1 && $rs <= $write_port->{MAX_DEF}+1)
{
                                my $waddr = wfield("addr", $write_port,
$ws);
                                my $raddr = rfield("addr", $read_port,
$rs);
                                my $we = wfield("we", $write_port, $ws);
                                my $kill = "kill_C$ws";
                                my $bypass = "bypass_result_$read_port-
>{NAME}_C$rs\_$write_port->{NAME}_C$ws";
                                print "    wire $bypass = ($waddr ==
$raddr) & $we & ~$kill;\n";
                            }
                          }
                      }
                  }
                  print "\n";

                  for($rs = $read_port->{MIN_USE}-1; $rs <= $read_port-
>{MAX_USE}-1; $rs++) {
                    my $mux = rfield("mux", $read_port, $rs);
                    my $mux_result = rfield("mux_result", $read_port,
$rs);
```

```
            my $rdata = rfield("data", $read_port, $rs);
            my $rdata1 = rfield("data", $read_port, $rs+1);

            print "    // Read bypass for port $read_port->{NAME}
use " . ($rs+1) . "\n";
            if ($rs == $read_port->{MIN_USE} - 1) {
                my(@data, @sel);
                # bypass the results from the write pipeline(s)
                for($ws = $rs+1; $ws <= $rf->{MAX_LATENCY}+1;
$ws++) {

                  foreach $write_port (@{$rf->{WRITE_PORT}}) {
                      if ($ws > 1 && $rs <= $write_port-
>{MAX_DEF}+1) {

                        my $result = wfield("result",
$write_port, $ws);

                        my $bypass = "bypass_result_$read_port-
>{NAME}_C$rs\_$write_port->{NAME}_C$ws";
                        push(@data, $result);
                        push(@sel, $bypass);
                      }
                  }
                }

                # lowest priority is data from register file
                push(@data, $rdata);
                print "    wire $rf->{WORD_DECL} $mux_result;\n";
                inline_mux(\@data, \@sel, $rf->{WORD_SIZE},
$mux_result, "priority");
                $rdata = $mux_result;
            }


            # choose binary encoding for the data bypass mux
            # order stage 2 last, read data first
            my(@data, @sel, $ncode, %code);
            $ncode = 0;
            $code{$rdata} = $ncode++;
            for($ws = $rs+1; $ws <= $rf->{MAX_LATENCY}+1; $ws++)
{

                foreach $write_port (@{$rf->{WRITE_PORT}}) {
                  if ($rs <= $write_port->{MAX_DEF}+1) {
                      if ($ws != 2 && write_def($write_port,
$ws)) {

                        my $wdata = wfield("data", $write_port,
$ws);

                        $code{$wdata} = $ncode++;
                      }
                  }
                }
            }
            foreach $write_port (@{$rf->{WRITE_PORT}}) {
                if (write_def($write_port, 2)) {
                  my $wdata = wfield("data", $write_port, 2);
                  $code{$wdata} = $ncode++;
                }
            }
```

```
                    # build the priority-encoded bypass mux
                    for($ws = $rs+1; $ws <= $rf->{MAX_LATENCY}+1; $ws++)
 {
                        foreach $write_port (@{$rf->{WRITE_PORT}}) {
                          if ($rs <= $write_port->{MAX_DEF}+1) {
                              if (write_def($write_port, $ws)) {
                                my $wdata = wfield("data", $write_port,
$ws);

                                my $bypass = "bypass_data_$read_port-
>{NAME}_C$rs\_$write_port->{NAME}_C$ws";
                                push(@data, $wdata);
                                push(@sel, $bypass);
                              }
                              if ($ws > 1) {
                                my $bypass = "bypass_result_$read_port-
>{NAME}_C$rs\_$write_port->{NAME}_C$ws";
                                push(@data, $rdata);
                                push(@sel, $bypass);
                              }
                          }
                        }
                    }
                    push(@data, $rdata);

                    print "    wire $rf->{WORD_DECL} $mux;\n";
                    inline_mux(\@data, \@sel, $rf->{WORD_SIZE}, $mux,
"priority", \%code);
                    print "    xtdelay1 #($rf->{WORD_SIZE})
i$rdata1($rdata1, $mux, clk);\n";
                    print "\n";
                  }
              }
            }

            print "    assign Stall_R =\n";
            foreach $write_port (@{$rf->{WRITE_PORT}}) {
              foreach $read_port (@{$rf->{READ_PORT}}) {
                  for($s = 1; $s <= $write_port->{MAX_DEF}-1; $s++) {
                    my($waddr) = wfield("addr", $write_port, $s);
                    my($raddr) = rfield("addr", $read_port, 0);
                    print "          (($waddr == $raddr) & (\n";
                    for($i = 1; $i <= $write_port->{MAX_DEF} - $s; $i++)
 {

                        my($use) = rfield("use$i", $read_port, 0);
                        print "              ($use & (";
                        for($j = $i+$s; $j <= $write_port->{MAX_DEF};
$j++) {

                          my($ns_def) = wfield("ns_def$j", $write_port,
$s);

                          print "$ns_def";
                          if ($j != $write_port->{MAX_DEF}) {
                              print " | ";
                          }
                        }
                        print "))";
                        if ($i == $write_port->{MAX_DEF} - $s) {
                          print ")) |\n";
```

```
                    } else {
                      print "  |\n";
                    }
                  }
                }
              }
            }
            print "        1'b0;\n";
            print "\n";


    ####################################################################
    ####
            #   Drop the core-cell

    ####################################################################
    ####
            if ($main::verify) {
              print "    // verification register file core -- hack\n";
              my $last;
              foreach $write_port (@{$rf->{WRITE_PORT}}) {
                  my $data = wfield("result", $write_port, $write_port-
    >{MAX_DEF}+1);
                  my $we = wfield("ns_we", $write_port, $write_port-
    >{MAX_DEF}+1);
                  my $tmp = wfield("tmp", $write_port, $write_port-
    >{MAX_DEF}+1);
                  print "    wire $rf->{WORD_DECL} $tmp;\n";
                  print "    xtenflop #($rf->{WORD_SIZE}) x$tmp($tmp,
    $data, $we, clk);\n";
                  $last = $tmp;
              }
              foreach $read_port (@{$rf->{READ_PORT}}) {
                  my $data = rfield("data", $read_port, $read_port-
    >{MIN_USE}-1);
                  print "    xtflop #($rf->{WORD_SIZE}) x$data($data,
    $last, clk);\n";
              }
            } else {
              print "    // register file core\n";
              init_print_break(8);
              my $r = @{$rf->{READ_PORT}};
              my $w = @{$rf->{WRITE_PORT}};
              my $n = $rf->{MIN_HEIGHT};
              my $module = "xtregfile_${r}R${w}W_${n}";
              if (! $rf->{USE_LATCHES}) {
                  $module .= "_FF";
              }
              print_break("    $module #($rf->{WORD_SIZE}) icore(");
              foreach $read_port (@{$rf->{READ_PORT}}) {
                  my $data = rfield("data", $read_port, $read_port-
    >{MIN_USE} - 1);
                  print_break("$data, ");
                  if ($rf->{ADDR_SIZE} > 0) {
                    my $addr = rfield("addr", $read_port, $read_port-
    >{MIN_USE} - 1);
                    print_break("$addr, ");
```

```
                }
            }
            foreach $write_port (@{$rf->{WRITE_PORT}}) {
                my $data = wfield("result", $write_port, $write_port-
>{MAX_DEF}+1);
                print_break("$data, ");
                if ($rf->{ADDR_SIZE} > 0) {
                  my $addr = wfield("addr", $write_port, $write_port-
>{MAX_DEF}+1);
                  print_break("$addr, ");
                }
                my $we = wfield("ns_we", $write_port, $write_port-
>{MAX_DEF}+1);
                print_break("$we, ");
            }
            if ($rf->{USE_LATCHES} && $rf->{TEST_TRANSPARENT_LATCHES})
{
                print_break("TMode, ");
            }
            print_break("clk);\n");
        }

        print "endmodule\n";
    }


    sub set_def {
        my($rf) = @_;
        my($def, $s, $w, $read_port, $write_port, $field, $width,
$data_size, $addr_size);

    #    $def->{Kill_E} = {SIZE => 1, DIR => "in", DEFAULT => "0" };
        $def->{KillPipe_W} = {SIZE => 1, DIR => "in", DEFAULT => "0"
};
        $def->{Stall_R} = {SIZE => 1, DIR => "out" };

        foreach $read_port (@{$rf->{READ_PORT}}) {
          $data_size = $read_port->{MAX_WIDTH};
          $addr_size = $rf->{HI_ADDR_SIZE};

          $field = rfield("addr", $read_port, 0);
          $def->{$field} = { SIZE => $addr_size, DIR => "in", DEFAULT
=> "x"};

          foreach $s (@{$read_port->{USE}}) {
              $field = rfield("use$s", $read_port, 0);
              $def->{$field} = { SIZE => 1, DIR => "in", DEFAULT =>
"0"};

              $field = rfield("data", $read_port, $s);
              $def->{$field} = { SIZE => $data_size, DIR => "out" };
          }

          foreach $width (@{$read_port->{WIDTH}}) {
              $field = rfield("width$width", $read_port, 0);
```

```
                    $def->{$field} = { SIZE => 1, DIR => "in", DEFAULT =>
"0"};
                  }
                }

                foreach $write_port (@{$rf->{WRITE_PORT}}) {
                  $data_size = $write_port->{MAX_WIDTH};
                  $addr_size = $rf->{HI_ADDR_SIZE};

                  $field = wfield("addr", $write_port, 0);
                  $def->{$field} = { SIZE => $addr_size, DIR => "in", DEFAULT
=> "x"};

                  foreach $s (@{$write_port->{DEF}}) {
                      $field = wfield("def$s", $write_port, 0);
                      $def->{$field} = { SIZE => 1, DIR => "in", DEFAULT =>
"0"};

                      foreach $w (@{$write_port->{WIDTH}}) {
                        $field = wfield("data$w", $write_port, $s);
                        $def->{$field} = { SIZE => $data_size, DIR => "in",
DEFAULT => "x"};
                      }
                  }

                  foreach $s (1 .. $write_port->{MAX_DEF}) {
                      if ($s <= &max(@{$write_port->{DEF}})) {
                        $field = wfield("wen", $write_port, $s);
                        $def->{$field} = { SIZE => 1, DIR => "in", DEFAULT =>
"x"};
                      }
                  }

                  foreach $width (@{$write_port->{WIDTH}}) {
                      $field = rfield("width$width", $write_port, 0);
                      $def->{$field} = { SIZE => 1, DIR => "in", DEFAULT =>
"0"};
                  }
                }

                return $def;
            };


            sub regfile_stall_write {
                my($rf, $time, $addr, $width) = @_;
                my($i);
                for($i = 0; $i < $width / $rf->{MIN_WIDTH}; $i++) {
                  $main::regfile_stall->{$time}->{$addr + $i} = 1;
                }
            }

            sub regfile_stall_read {
                my($rf, $time, $addr, $width) = @_;
                my($i, $stall);
                $stall = 0;
```

```
            for($i = 0; $i < $width / $rf->{MIN_WIDTH}; $i++) {
              $stall |= defined $main::regfile_stall->{$time}->{$addr +
$i};
            }
            return $stall;
        }

        sub regfile_write {
            my($rf, $time, $addr, $data, $width) = @_;
            my($i);
            for($i = 0; $i < $width / $rf->{MIN_WIDTH}; $i++) {
              $main::regfile->{$time}->{$addr + $i} =
                  ($data >> ($i * $rf->{MIN_WIDTH})) & ((1 << $rf-
>{MIN_WIDTH}) - 1);
            }
        }

        sub regfile_read {
            my($rf, $time, $addr, $width) = @_;
            my($t, $out_value, $i);
            $out_value = 0;
            for($i = 0; $i < $width / $rf->{MIN_WIDTH}; $i++) {
              my($value);
              for($t = $time; $t >= 0; $t--) {
                  $value = $main::regfile->{$t}->{$addr + $i};
                  if (defined $value) {
                    last;
                  }
              }
              if (! defined $value) {
                  die "regfile_read: time=$time addr=$addr value
undefined";
              }
              $out_value |= $value << ($i * $rf->{MIN_WIDTH});
            }
            return $out_value;
        }


        sub init_field {
            my($rf, $time) = @_;
            my($field, $default, $size, $dir, $value);
            foreach $field (keys(%{$rf->{SIGNALS}})) {
              my($info) = $rf->{SIGNALS}->{$field};
              $default = $info->{DEFAULT};
              $size = $info->{SIZE};
              $dir = $info->{DIR};
              if ($dir eq "in" && $size > 0) {
                  if ($default eq "0") {
                    $value = 0;
                  } elsif ($default eq "x") {
                    $value = $size . "'b" . ('x' x $size);
                  } else {
                    die "Bad init field in $field\n";
                  }
                  add_field($rf, $time, $field, $value);
```

```
        } elsif ($field eq "Stall_R") {
            add_field($rf, $time, $field, "1:0");
        }
      }
  }


  sub add_field {
      my($rf, $time, $field, $value) = @_;
      my($info) = $rf->{SIGNALS}->{$field};
      die "add_field: field \"$field\" not found" if ! defined
$info;
      return if $info->{SIZE} == 0;
      if (! defined $main::vector->{$time}) {
        $main::vector->{$time} = { };
        init_field($rf, $time);
      }
      $main::vector->{$time}->{$field} = $value;
  }



  sub make_view_pipeline_register_cell {
      my($rf) = @_;
      my(@iolist, $read_port, $s, $w, $s, $write_port, $module);

      foreach $read_port (@{$rf->{READ_PORT}}) {
        foreach $s (@{$read_port->{USE}}) {
            my($data) = rfield("data", $read_port, $s);
            my($decl) = "[" . ($read_port->{MAX_WIDTH} - 1) .
":0]";
            push(@iolist, "$data, ");
            print TEST "    wire $decl $data;\n";
        }
        # don't need an address for a single word register file
        if ($rf->{HI_ADDR_SIZE} > 0) {
            my($addr) = rfield("addr", $read_port, 0);
            my($decl) = "[" . ($rf->{HI_ADDR_SIZE} - 1) . ":0]";
            push(@iolist, "$addr, ");
            print TEST "    reg $decl $addr;\n";
        }
        foreach $w (@{$read_port->{WIDTH}}) {
            my($width) = rfield("width$w", $read_port, 0);
            push(@iolist, "$width, ");
            print TEST "    reg $width;\n";
        }
        foreach $s (@{$read_port->{USE}}) {
            my($use) = rfield("use$s", $read_port, 0);
            push(@iolist, "$use, ");
            print TEST "    reg $use;\n";
        }
      }

      foreach $write_port (@{$rf->{WRITE_PORT}}) {
        # don't need an address for a single word register file
        if ($rf->{HI_ADDR_SIZE} > 0) {
            my($addr) = wfield("addr", $write_port, 0);
```

```
                my($decl) = "[" . ($rf->{HI_ADDR_SIZE} - 1) . ":0]";
                push(@iolist, "$addr, ");
                print TEST "    reg $decl $addr;\n";
            }
            foreach $w (@{$write_port->{WIDTH}}) {
                my($width) = rfield("width$w", $write_port, 0);
                push(@iolist, "$width, ");
                print TEST "    reg $width;\n";
            }
            foreach $s (@{$write_port->{DEF}}) {
                my($def) = wfield("def$s", $write_port, 0);
                push(@iolist, "$def, ");
                print TEST "    reg $def;\n";
            }
            foreach $w (@{$write_port->{WIDTH}}) {
                foreach $s (@{$write_port->{DEF}}) {
                  my($data) = wfield("data$w", $write_port, $s);
                  my($decl) = "[" . ($w - 1) . ":0]";
                  push(@iolist, "$data, ");
                  print TEST "    reg $decl $data;\n";
                }
            }
            foreach $s (1 .. $write_port->{MAX_DEF}) {
                if ($s <= &max(@{$write_port->{DEF}})) {
                  my($wen) = wfield("wen", $write_port, $s);
                  push(@iolist, "$wen, ");
                  print TEST "    reg $wen;\n";
                }
            }
        }

        push(@iolist, "Kill_E, ");
        print TEST "//   reg Kill_E;\n";

        push(@iolist, "KillPipe_W, ");
        print TEST "    reg KillPipe_W;\n";

        push(@iolist, "Stall_R, ");
        print TEST "    wire Stall_R;\n";

        push(@iolist, "clk);\n");
        print TEST "    reg clk;\n";

        print TEST "    $rf->{NAME} i0(";
        print TEST join('', @iolist);
        print TEST "\n";
    }


    sub print_vector {
        my($rf) = @_;
        my($time, $size, $value, $width, $last_value, $mask, $addr,
$dir, $field);
        my($max_time) = max(keys(%$main::vector));

        print TEST "module driver;\n";
```

```
            make_view_pipeline_register_cell($rf);

            print TEST "    initial begin\n";
            print TEST "        #2 ;\n";

            for($time = 0; $time <= $max_time; $time++) {
              print TEST "\n";
              print TEST "\n";
              print TEST "    // time: $time\n";
              foreach $field (sort (keys(%{$main::vector->{$time}}))) {
                  $dir = $rf->{SIGNALS}->{$field}->{DIR};
                  next if $dir ne "in";
                  $value = $main::vector->{$time}->{$field};
                  $last_value = $main::vector->{$time-1}->{$field};
                  if ($time == 0 || ! defined $last_value || $value ne
$last_value) {
                    print TEST "    $field = $value;\n";
                  }
              }

              print TEST "    #5;\n";

              if (defined $main::print_vector{$time}) {
                  print TEST "
\$display(\"$main::print_vector{$time}\");\n";
              }
              foreach $field (sort (keys(%{$main::vector->{$time}}))) {
                  $dir = $rf->{SIGNALS}->{$field}->{DIR};
                  next if $dir ne "out";
                  ($width, $value) = split(':', $main::vector->{$time}-
>{$field});
                  if ($field ne "Stall_R") {
                    $field = $field . "[" . ($width - 1) . ":0]";
                  }
                  print TEST "    if ($field != $value) begin\n";
                  print TEST "        \$display(\"FAIL! %d $field %d
$value\", \$time, $field);\n";
                  print TEST "    end\n";
              }
              print TEST "    #5 ;\n";
            }

          print TEST "    end\n";

          print TEST "xtflop #(1) dummy(Kill_E, Stall_R, clk);\n\n";
          print TEST "initial begin clk = 1; end\n";
          print TEST "always begin #5 clk = ~clk; end\n\n";
          print TEST "always begin #(" . ($max_time+10) . "*10)
\$finish; end\n";
          print TEST "endmodule\n";
      }
```

```
$main::try_kill = 0;
$main::time = 0;
$main::nop_count = 0;

sub inst {
    my($rf, $arg, $kill) = @_;
    my($write_port, $read_port, $write_port_num, $read_port_num);
    my($i, $arg_print, $op, $field, @operand, $stall, $port,
$addr, $width, $data, $def, $use, $time);

    $time = $main::time++;
    @operand = split(' ', $arg);
    $arg_print = "";

    # check for stall on any read port
    $stall = 0;
    foreach $op (@operand) {
      next if substr($op, 0, 1) eq ">";
      ($port, $addr, $use, $width) = split('-', $op);
      $stall |= regfile_stall_read($rf, $time + $use, $addr,
$width);
    }
    if ($stall) {
      add_field($rf, $time, "Stall_R", "1:1");
    }

    # if there is no stall, try a random killpipe when this
    # instruction reaches W
    if ($main::try_kill && $kill && $stall == 0 && int(rand(20))
== 0) {
      $main::nop_count = 4;
      add_field($rf, $time + 3, "KillPipe_W", 1);
      $arg_print .= sprintf("%-10s ", "Kill!");
    }

    # process the read(s)
    foreach $op (@operand) {
      next if substr($op, 0, 1) eq ">";

      ($port, $addr, $use, $width) = split('-', $op);
      $read_port = $rf->{READ_PORT}->[$port];

      $field = rfield("addr", $read_port, 0);
      add_field($rf, $time, $field, $addr);

      $field = rfield("use$use", $read_port, 0);
      add_field($rf, $time, $field, 1);

      $field = rfield("width$width", $read_port, 0);
      add_field($rf, $time, $field, 1);

      $data = regfile_read($rf, $time, $addr, $width);
      if (! $stall) {
          $field = rfield("data", $read_port, $use);
          add_field($rf, $time + $use, $field, "$width:$data");
      }
```

```
            $arg_print .= sprintf("%-20s", "$op=$data ");
          }

          # process the write(s)
          foreach $op (@operand) {
            next if substr($op, 0, 1) ne ">";
            ($port, $addr, $def, $width) = split('-', substr($op,1));
            $write_port = $rf->{WRITE_PORT}->[$port];

            $field = wfield("addr", $write_port, 0);
            add_field($rf, $time, $field, $addr);

            $field = wfield("def$def", $write_port, 0);
            add_field($rf, $time, $field, 1);

            $field = wfield("width$width", $write_port, 0);
            add_field($rf, $time, $field, 1);

            $field = wfield("data$width", $write_port, $def);
            $data = int(rand(pow2($width)));
            add_field($rf, $time + $def, $field, $data);

            if (! $stall) {
                for($i = 1; $i <= $def; $i++) {
                  $field = wfield("wen", $write_port, $i);
                  add_field($rf, $time + $i, $field, 1);
                  regfile_stall_write($rf, $time + $i, $addr, $width);
                }
                if ($main::nop_count == 0) {
                  regfile_write($rf, $time, $addr, $data, $width);
                }
            }

            $arg_print .= sprintf("%-20s", "$op=$data ");
          }
          if ($main::nop_count > 0) {
            $main::nop_count--;
          }

          $main::print_vector{$time} = sprintf("%4d: %d %s", $time,
$stall, $arg_print);

          # replay the instruction on a stall
          if ($stall) {
            inst($rf, $arg, $kill);
          }
      }
```

```
    sub test_view_pipeline_regfile {
        my($rf) = @_;
        my($i, $num, $port, $addr, $use, $def, $width, $op,
$read_port, $write_port);

        # write each address using max write-width, min def, min port
#
        $write_port = $rf->{WRITE_PORT}->[0];
        $width = $write_port->{WIDTH}->[$#{@{$write_port->{WIDTH}}}];
        for($addr = 0; $addr < $rf->{SIZE} / $width; $addr++) {
          $a = $addr * $width / $rf->{MIN_WIDTH};
          $def = @{$write_port->{DEF}}[0];
          $op = ">0-$a-$def-$width";
          inst($rf, $op, 0);
        }

        # flush the pipeline
        for($i = 0; $i < 10; $i++) {
          inst($rf, "", 0);
        }

        # read each address using each read-width, each use, each
port
        $port = 0;
        foreach $read_port (@{$rf->{READ_PORT}}) {
          foreach $use (@{$read_port->{USE}}) {
              foreach $width (@{$read_port->{WIDTH}}) {
                for($addr = 0; $addr < $rf->{SIZE} / $width; $addr++)
{
                    $a = $addr * $width / $rf->{MIN_WIDTH};
                    $op = "$port-$a-$use-$width";
                    inst($rf, $op, 0);
                }
              }
          }
          $port++;
        }

        while ($main::time < $rf->{NUM_TEST_VECTOR} - 10) {
          $op = "";
          for($port = 0; $port < @{$rf->{READ_PORT}}; $port++) {
              if (int(rand(8)) != 0) {
                $read_port = @{$rf->{READ_PORT}}[$port];
                $num = @{$read_port->{WIDTH}};
                $width = $read_port->{WIDTH}->[int(rand($num))];
                $addr = int(rand($rf->{SIZE} / $width)) * $width /
$rf->{MIN_WIDTH};
                $num = @{$read_port->{USE}};
                $use = $read_port->{USE}->[int(rand($num))];
                $op .= " $port-$addr-$use-$width";
              }
          }

          for($port = 0; $port < @{$rf->{WRITE_PORT}}; $port++) {
              if (int(rand(8)) != 0) {
                $write_port = @{$rf->{WRITE_PORT}}[$port];
```

```
                    $num = @{$write_port->{WIDTH}};
                    $width = $write_port->{WIDTH}->[int(rand($num))];
                    $addr = int(rand($rf->{SIZE} / $width)) * $width /
$rf->{MIN_WIDTH};
                    $num = @{$write_port->{DEF}};
                    $def = $write_port->{DEF}->[int(rand($num))];
                    $op .= " >$port-$addr-$def-$width";
                  }
              }
              inst($rf, $op, 1);
            }

            # flush the pipeline
            for($i = 0; $i < 10; $i++) {
              inst($rf, "", 0);
            }

            print_vector($rf);
        }



        my($rf, $rf_all, $i, $TEST, $usage, $ret);
        srand 1;

        # default values
        $main::verify = 0;
        $usage = <<EOF;
        usage: $0 [options]
        EOF
        # parse the command line
        $ret = GetOptions(
            "verify" => \$main::verify,
        );
        if (! $ret) {
            print "$usage";
            exit 1;
        }

        $rf_all = '$rf_all = [ ' . join(' ', <>) . '] ;';
        eval($rf_all) || die "Syntax error in input description";

        for($i = 0; $i < @$rf_all; $i++) {
            $rf = $rf_all->[$i];

            derive_constants($rf);
            $rf->{SIGNALS} = set_def($rf);

            write_regfile($rf);
            print "\n\n";
            write_regfile_bank($rf);
            print "\n\n";

            if (defined $rf->{TEST_FILENAME}) {
              $TEST = $rf->{TEST_FILENAME};
              open(TEST, ">$TEST") || die "Can't open $TEST: $!";
              test_view_pipeline_regfile($rf);
```

```
            close TEST;
        }
}
```

APPENDIX C

10. Instruction Latency and Throughput

10.1 Introduction

This book describes the Xtensa Instruction Set Architecture (ISA). The ISA is defined independently of its various implementations, so that software that targets the ISA will run on any its implementations. The ISA includes features are not required by some of its implementations, but which will be important to include in software written today if it is to work on future implementations (e.g. using MEMW and EXCW). While correct software must adhere to the ISA and not to the specifics of any of its implmentations, it is sometimes important to know the details of an implementation for performance reasons, such as scheduling instructions to avoid pipeline delays. This chapter provides an overview of performance modeling, and then provides detailed information for each implementation designed to date.

10.2 Processor Performance Terminology and Modeling

It is important to have a model of processor performance for both code generation and simulation. However, the interactions of multiple instructions in a processor pipeline can be complex. It is common to simplify and describe pipeline and cache performance separately even though they may interact, as the information is used in different stages of compilation or coding. We adopt this approach, and then separately describe some of the interactions. It is also common to describe the pipelining of instructions with *latency* (the time an instruction takes to produce its result after it receives its inputs) and *throughput* (the time an instruction delays other instructions independent of operand dependencies) numbers, but this cannot accommodate some situations. Therefore we adopt a slightly more complicated, but more accurate model. This model focuses on predicting when one instruction *issues* relative to other instructions. An instruction issues when all of its data inputs are available and all the necessary hardware functional units are available for it. Issue is the point at which computation of the instruction's results begins.

Instead of using a per-instruction latency number, instructions are modeled as taking their operands in various pipeline stage numbers, and producing results in various pipeline stage numbers. When instruction IA writes X (either an explicit operand or implicit state register) and instruction IB reads X then instruction IB depends on IA.[1] If instruction IA produces X in stage SA, and instruction IB uses X in stage SB, then instruction IB can issue no earlier than $D = \max(SA - SB, 0)$ cycles after IA issued. This is illustrated in

1. This situation is called a "read after write" dependency. Other possible operand dependencies familiar to coders are "write after write" and "write after read," but these have no pipeline performance implications in any existing Xtensa processor implementation, and thus are not discussed further here.

Figure 23. If the processor reaches IB earlier than D cycles after IA, it generally delays IB's issue into the pipeline until D cycles have elapsed. When the processor delays an instruction because of a pipeline interaction, we call it an "interlock." For a few special dependencies (primarily those involving the special registers controlling exceptions, interrupts, and memory management) the processor does not interlock. These situations are called "hazards." For correct operation, code generation must insert xSYNC instructions to avoid hazards by delaying the dependent instruction. The xSYNC series of instructions is designed to accomplish this delay in an implementation-independent manner.

When we describe an instruction as making one of its values available at the beginning of some stage, this refers to when the computation is complete, and not necessarily the time that the actual processor state is written. It is usual to delay the state write until at least the point at which the instruction is committed (i.e. cannot be aborted by its own or an earlier instruction's exception). In some implementations the state write is delayed still further to satisfy resource constraints. However, the delay in writing the actual processor state is usually invisible; most processors will detect the use of an operand that has been produced by one instruction and is being used by another even though the processor state has not been written, and forward the required value from one pipeline stage to the other. This operation is called *bypass*.

Instructions may be delayed in a pipeline for reasons other than operand dependencies. The most common situation is for two or more instructions to require a particular piece of the processor's hardware (called a "functional unit") to execute. If there are fewer copies of the unit than instructions that need to use the unit in a given cycle, the processor must delay some of the instructions to prevent the instructions from interfering with each other. For example, a processor may have only one read port for its data cache. If instruction IC uses this read port in its stage 4 and instruction ID uses the read port in its stage 3, then it would not be possible to issue IC in cycle 10 and ID in cycle 11, because they would both need to use the data cache read port in cycle 14. Typically the processor would delay ID's issue into the pipeline by one cycle to avoid conflict with IC.

Modern processor pipeline design tends to avoid the use of functional units in varying pipeline stages by different instructions and to fully pipeline functional unit logic, which means that most instructions would conflict with each other on a shared functional unit only if they issued in the same cycle. However, there are usually still a small number of cases in which a functional unit is used for several cycles. For example, floating-point or integer division may iterate for several cycles in a single piece of hardware. In this case, once a divide has started, it is not possible to start another divide until the first has left the iterative hardware. This is illustrated in Figure 24.

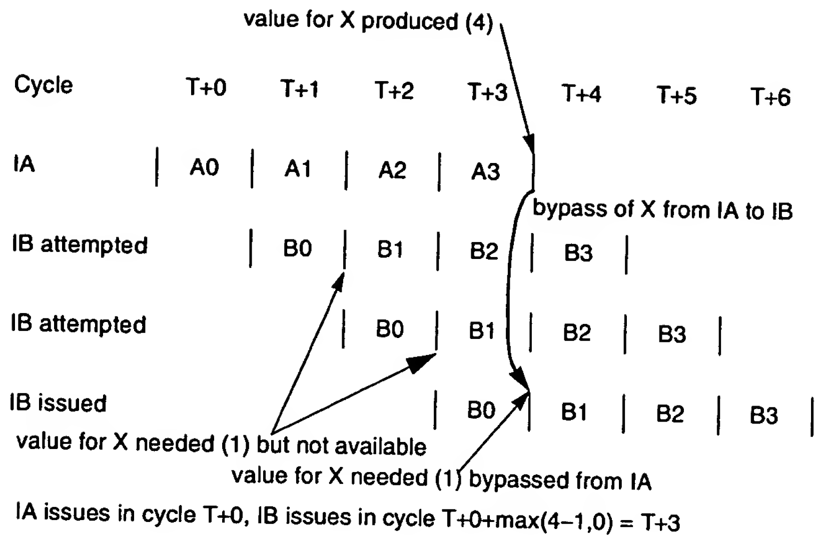
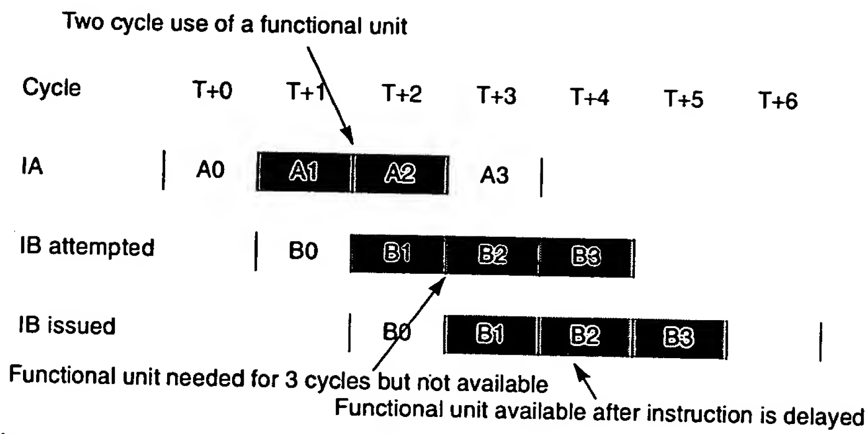


Figure 23. Instruction Operand Dependency Interlock

10.3 Instruction Latency and Throughput Table Description

Subsequent sections give tables that describe each Xtensa implementation's pipeline characteristics. The table has one row per instruction, and columns that give pipeline stage numbers for the input and output of both explicit operands and implicit operands, and a final column that gives the functional unit reservations required by that instruction. The preface to the table gives the number of copies of each functional unit. This information is generally sufficient to predict the pipeline delays that a series of instructions will experience. As described earlier, any pair of instructions A and B with a read-after-write dependency on X must be separated in the pipeline by at least max(AXoutstage – BXinstage, 0) cycles. In addition, the separation must be such that the functional unit reservations of B when added to the reservations of all prior instructions must not exceed the number of copies of any functional unit on any cycle.

Output stage numbers represent the stage number in which the result is available for use as an input operand (e.g. via bypass). This is likely different from the stage number in which the actual processor state is written (this typically happens one or more stages later). For example, an instruction with input specification "`as` 1" and output specification "`ar` 2" requires its `as` input at the beginning of stage 1, and makes its `ar` output available at the beginning of stage 2. The latency is 1 cycle (2 – 1 = 1).



IA issues at T+0 and reserves the functional unit in cycles T+1 and T+2
IB tries to issue at T+1 and reserve the unit in T+2..T+4, but is blocked by IA's T+2 reservation
IB is retried and issues in cycle T+2, thereby avoiding IA's reservations

Figure 24. Functional Unit Interlock

Branch instructions have no explicit output operand, but they do produce an instruction address that is required by the instruction fetch unit of the processor. Thus the time between a taken branch and its target may be modeled by the same procedure by adding a target pseudo-output to the branch instructions in the table. For processors with branch prediction, this represents the stage at which a branch mispredict is detected. Most processors without branch prediction have no penalty for untaken branches.

Instructions defined in TIE may optionally specify equivalent input and output stage numbers for instructions via the `schedule` declaration. If no `schedule` declaration is given, an implementation-defined default is used. For most implementations this default gives TIE instructions the same timing characteristics as the Xtensa ADD instruction.

The tables do not attempt to represent the effects of accessing instructions or data operands that miss in the cache.

10.4 Xtensa T1000

T1000 is Tensilica's first implementation of the Xtensa Instruction Set Architecture. This material covers the 2.0 release of Xtensa T1000. The 1.5 release is similar in nearly all aspects covered here. Please consult product datasheets for the differences between the 1.5 and 2.0 releases.

Xtensa T1000 uses a five-stage pipeline capable of executing at most one instruction per cycle. The pipeline stages are described in Table 136. The first stage, `I`, is partially decoupled from the next, `R`, and `R` is partially decoupled from the last three stages, `E`, `M`,

and W, which operate in lock-step. If an interlock condition is detected in the R stage then in the next cycle the instruction is retried in R and a no-op is sent on to the E stage. If an instruction is held in R, then the word fetched in I is captured in a buffer, and on the following cycle I performs no operation.

Table 136. Xtensa T1000 Pipeline

Name	Description
I	Instruction Cache/RAM/ROM access Instruction Cache tag comparison Instruction alignment
R	AR register file read Instruction decode, interlocking, and bypass Instruction Cache miss recognition
E	Execution of most ALU-type instructions (ADD, SUB, etc.) Virtual address generation for load and store instructions Branch decision and address selection
M	Data Cache/RAM/ROM access for load and store instructions Data Cache tag comparison Data Cache miss recognition Load data alignment
W	State writes (e.g. AR register file write)

The three primary implications of the Xtensa T1000 pipeline are shown in Figure 25.

- Instructions that depend on an ALU result can execute with no delay because their result is available at the beginning of M and is needed at the beginning of E by the dependent instruction.
- Instructions that depend on load instruction results must issue two cycles after the load because the load result is available at the beginning of its W stage and is needed at the beginning of E by the dependent instruction. For best performance code generation should put an independent instruction in between the load and any instruction that uses the load result.
- Finally, the branch decision occurs in E, and for taken branches must affect the I stage of the target fetch, and so there are two fetched fall-through instructions that are killed on taken branches.

The processor uses 32-bit aligned fetches from the Instruction Cache/RAM/ROM. If the target of a branch is an instruction that crosses a 32-bit boundary, then two fetches will be required before the entire instruction is available, and so the target instruction will begin 3 cycles after the branch instead of 2. For best performance, code generation should align 24-bit targets of frequently taken branches on 0 or 1 mod 4 byte boundaries, and 16-bit targets on 0, 1, or 2 mod 4 byte boundaries.

The processor avoids overflowing its write buffer by interlocking in the `R` stage on stores when the write buffer is full or might become full from stores in the `E` and `M` stages.

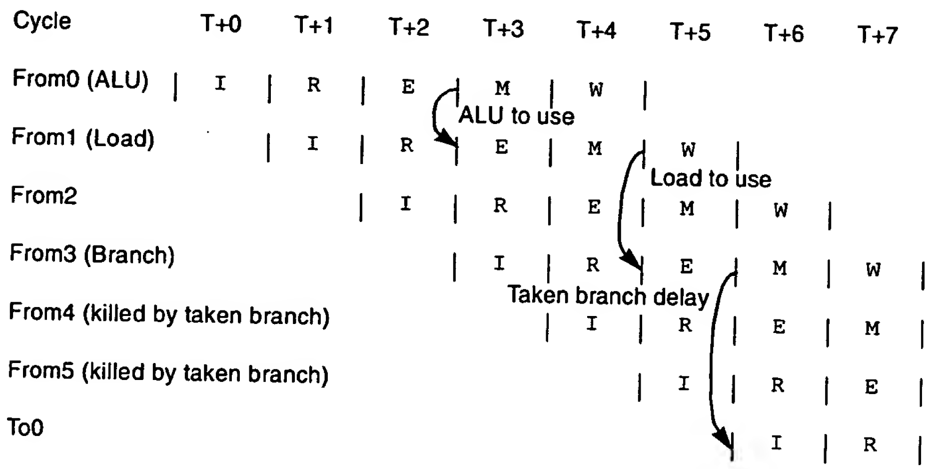


Figure 25. Xtensa T1000 Pipeline Effects

The windowed register subroutine return instructions, `RETW` and `RETW.N`, both use `AR[0]` from the register file without bypassing of pending writes from the pipeline, and so the processor interlock is much longer. This is approximated in Table 138 by specifying a negative stage number for the `AR[0]` read stage. This table entry is approximate because it still depends on the stage number of the instruction writing `AR[0]` whereas the actual processor interlock is independent of this (e.g. both `ADDI` and `L32I` to `AR[0]` are separated from `RETW` by 4 cycles, even though `ADDI`'s result is normally available earlier for bypass than `L32I`'s – both write the `AR` register file in the `W` stage).

Both instruction and data cache misses wait for the processor's write buffer (pending stores to the system) to empty before initiating the cache refill read.

Data cache misses are recognized in the `M` stage of the pipeline and are handled by flushing the `I` to `M` stages of the pipeline, including the load instruction, and then after the data cache line is refilled, the load instruction is refetched and reexecuted starting with the `I` stage. Data cache misses that are recognized after an instruction cache refill is initiated first wait for the instruction cache refill, and then initiate the data cache refill read.

Instruction cache misses are recognized in the `R` stage of the pipeline and are processed in parallel with execution of the instructions in the `E`, `M`, and `W` stages. If an instruction cache miss is recognized in `R` in the same cycle as a data cache miss in `M`, the data cache miss is serviced first, and then the instruction cache miss. While an instruc-

tion cache miss is being held waiting for the write buffer to empty, the rest of the pipeline continues to advance, which may allow a subsequent data cache miss to be detected in M; in this case the data cache miss will also take priority.

If an instruction cache miss occurs fetching the instruction immediately following a conditional branch instruction, the instruction cache miss recognition is delayed one cycle to determine whether the conditional branch is taken or not. If it is taken, the instruction cache miss is not serviced. If the branch is not taken, there will be a one cycle delay in the start of the instruction cache refill of the fall-through instruction cache line.

Instructions that reference the Instruction Cache/RAM/ROM as data, i.e. LICT, LICW, SICT, SICW, IHI, and III, operate by flushing the pipeline from the following instruction, performing the operation, and then refetching the following instruction.

10.4.1 Xtensa T1000 Instruction Latency and Throughput Tables

Xtensa T1000's single-issue restriction is equivalent to adding one use of a Fetch unit in stage 0 to every instruction. This is not listed in Table 138. Hardware that could be considered as a separate functional unit is not listed if it cannot cause additional delay. For example, the integer adder used by ADD, SUB, etc. could be considered a separate functional unit, but the delays that it would introduce are a subset of those caused by the Fetch functional unit, and so it is not listed. The relevant functional units are listed in Table 137.

In Table 138 stage 0 corresponds to Xtensa T1000's R stage, stage 1 to the E stage, etc.

The default stages for TIE instructions if no schedule declaration is specified is for inputs to be read at the beginning of stage 1 and outputs to be available at the beginning of stage 2 (i.e. similar to the ADD instruction).

Table 137. Xtensa T1000 Functional Units

Name	Copies	Description
Fetch	1	Instruction fetch unit delivers at most one instruction per cycle. A use of this unit in stage 0 is implicit in every instruction and does not appear in Table 138.
DataCache	1	The processor's data cache is generally accessed in stage 2, but the data cache invalidation instructions (e.g. DHI) conditionally access the cache in stage 3 as well.
WSR	1	This unit is used to model the interaction of WSR and the xSYNC instructions.

Table 138. Xtensa T1000 Instruction Pipelining

Instruction Mnemonic	Operand Stages		Implicit State Stages		Reservations
	In	Out	In	Out	
ABS	at 1	ar 2	—	—	—
ADD	as 1, at 1	ar 2	—	—	—
ADD.N	as 1, at 1	ar 2	—	—	—
ADDI	as 1	at 2	—	—	—
ADDI.N	as 1	ar 2	—	—	—
ADDMI	as 1	at 2	—	—	—
ADDX2	as 1, at 1	ar 2	—	—	—
ADDX4	as 1, at 1	ar 2	—	—	—
ADDX8	as 1, at 1	ar 2	—	—	—
AND	as 1, at 1	ar 2	—	—	—
ALL4	bs 1	bt 2	—	—	—
ALL8	bs 1	bt 2	—	—	—
AND	as 1, at 1	ar 2	—	—	—
ANDB	bs 1, bt 1	br 2	—	—	—
ANDBC	bs 1, bt 1	br 2	—	—	—
ANY4	bs 1	bt 2	—	—	—
ANY8	bs 1	bt 2	—	—	—
BALL	as 1, at 1	target 2	—	—	—
BANY	as 1, at 1	target 2	—	—	—
BBC	as 1, at 1	target 2	—	—	—
BBCI	as 1	target 2	—	—	—
BBS	as 1, at 1	target 2	—	—	—
BBSI	as 1	target 2	—	—	—
BEQ	as 1, at 1	target 2	—	—	—
BEQI	as 1	target 2	—	—	—
BEQZ	as 1	target 2	—	—	—
BEQZ.N	as 1	target 2	—	—	—
BF	bs 1	—	—	—	—
BGE	as 1, at 1	target 2	—	—	—
BGEI	as 1	target 2	—	—	—
BGEU	as 1, at 1	target 2	—	—	—

1. The RETW and RETW.N instructions lack bypass for the AR[0] value. This is approximated by specifying that they use AR[0] in a much earlier stage (–2). See Section 10.4 on page 528.

Table 138. Xtensa T1000 Instruction Pipelining (continued)

Instruction Mnemonic	Operand Stages		Implicit State Stages		Reservations
	In	Out	In	Out	
BGEUI	as 1	target 2	—	—	—
BGEZ	as 1	target 2	—	—	—
BLT	as 1, at 1	target 2	—	—	—
BLTI	as 1	target 2	—	—	—
BLTU	as 1, at 1	target 2	—	—	—
BLTUI	as 1	target 2	—	—	—
BLTZ	as 1	target 2	—	—	—
BNALL	as 1, at 1	target 2	—	—	—
BNE	as 1, at 1	target 2	—	—	—
BNEI	as 1	target 2	—	—	—
BNEZ	as 1	target 2	—	—	—
BNEZ.N	as 1	target 2	—	—	—
BNONE	as 1, at 1	target 2	—	—	—
BREAK	—	—	—	—	—
BREAK.N	—	—	—	—	—
BT	bs 1	—	—	—	—
CALL0	—	target 2	—	AR[0] 2	—
CALL4	—	target 2	—	AR[4] 2	—
CALL8	—	target 2	—	AR[8] 2	—
CALL12	—	target 2	—	AR[12] 2	—
CALLX0	as 1	target 2	—	AR[0] 2	—
CALLX4	as 1	target 2	—	AR[4] 2	—
CALLX8	as 1	target 2	—	AR[8] 2	—
CALLX12	as 1	target 2	—	AR[12] 2	—
CHK	*not implemented*				
CLAMPS	as 1	ar 2	—	—	—
CRC8	as 1, at 1	ar 2	—	—	—
DHI	as 1	—	—	—	DataCache 2 3
DHWB	*implemented as NOP*				
DHWBI	as 1	—	—	—	DataCache 2 3
DII	as 1	—	—	—	DataCache 2

1. The RETW and RETW.N instructions lack bypass for the AR[0] value. This is approximated by specifying that they use AR[0] in a much earlier stage (–2). See Section 10.4 on page 528.

Table 138. Xtensa T1000 Instruction Pipelining (continued)

Instruction Mnemonic	Operand Stages		Implicit State Stages		Reservations
	In	Out	In	Out	
DPFL	*not implemented – illegal instruction*				
DPFR	*implemented as NOP*				
DPFRO	*implemented as NOP*				
DPFW	*implemented as NOP*				
DPFWO	*implemented as NOP*				
DSYNC	—	—	—	—	WSR 1
ENTRY	as 1	as 2	—	—	—
ESYNC	—	—	—	—	WSR 1..2
EXCW	—	—	—	—	—
EXTUI	at 1	ar 2	—	—	—
IHI	as 1	—	—	—	Fetch 0..5
III	as 1	—	—	—	Fetch 0..5
INVAL	*not implemented*				
IPF	as 1	—	—	—	—
IPFL	*not implemented*				
ISYNC	—	—	—	—	Fetch 0..4
J	—	target 2	—	—	—
JX	as 1	target 2	—	—	—
L8UI	as 1	at 3	MEM 2	—	DataCache 2
L16SI	as 1	at 3	MEM 2	—	DataCache 2
L16UI	as 1	at 3	MEM 2	—	DataCache 2
L32AI	*not implemented*				
L32I	as 1	at 3	MEM 2	—	DataCache 2
L32I.N	as 1	at 3	MEM 2	—	DataCache 2
L32R	—	at 3	MEM 2	—	DataCache 2
L32SI	*not implemented*				
LDCT	as 1	at 3	—	—	DataCache 2
LDDEC	as 1	mw 3, as 2	MEM 2	—	DataCache 2
LDINC	as 1	mw 3, as 2	MEM 2	—	DataCache 2
LICT	as 1	at 3	—	—	Fetch 0..4
LICW	as 1	at 3	—	—	Fetch 0..4

1. The RETW and RETW.N instructions lack bypass for the AR[0] value. This is approximated by specifying that they use AR[0] in a much earlier stage (–2). See Section 10.4 on page 528.

Table 138. Xtensa T1000 Instruction Pipelining (continued)

Instruction Mnemonic	Operand Stages		Implicit State Stages		Reservations
	In	Out	In	Out	
LOOP	as 1	—	—	LBEG 2, LEND 2, LCOUNT 2	—
LOOPGTZ	as 1	target 2	—	LBEG 2, LEND 2, LCOUNT 2	—
LOOPNEZ	as 1	target 2	—	LBEG 2, LEND 2, LCOUNT 2	—
MAX	as 1, at 1	ar 2	—	—	—
MAXU	as 1, at 1	ar 2	—	—	—
MEMW	—	—	—	—	—
MIN	as 1, at 1	ar 2	—	—	—
MINU	as 1, at 1	ar 2	—	—	—
MOV.N	as 1	at 2	—	—	—
MOVEQZ	as 1, at 1	ar 2	—	—	—
MOVF	as 1, bt 1	ar 2	—	—	—
MOVGEZ	as 1, at 1	ar 2	—	—	—
MOVI	—	at 2	—	—	—
MOVI.N	—	as 2	—	—	—
MOVLTZ	as 1, at 1	ar 2	—	—	—
MOVNEZ	as 1, at 1	ar 2	—	—	—
MOVSP	as 1	at 2	—	—	—
MUL16S	as 1, at 1	ar 3	—	—	—
MUL16U	as 1, at 1	ar 3	—	—	—
MOVT	as 1, bt 1	ar 2	—	—	—
MUL.AA.HH	as 1, at 1	—	—	ACCLO 3, ACCHI 3	—
MUL.AA.HL	as 1, at 1	—	—	ACCLO 3, ACCHI 3	—
MUL.AA.LH	as 1, at 1	—	—	ACCLO 3, ACCHI 3	—
MUL.AA.LL	as 1, at 1	—	—	ACCLO 3, ACCHI 3	—

1. The RETW and RETW.N instructions lack bypass for the AR[0] value. This is approximated by specifying that they use AR[0] in a much earlier stage (–2). See Section 10.4 on page 528.

Table 138. Xtensa T1000 Instruction Pipelining (continued)

Instruction Mnemonic	Operand Stages		Implicit State Stages		Reservations
	In	Out	In	Out	
MUL.AD.HH	as 1, my 1	—	—	ACCLO 3, ACCHI 3	—
MUL.AD.HL	as 1, my 1	—	—	ACCLO 3, ACCHI 3	—
MUL.AD.LH	as 1, my 1	—	—	ACCLO 3, ACCHI 3	—
MUL.AD.LL	as 1, my 1	—	—	ACCLO 3, ACCHI 3	—
MUL.DA.HH	mx 1, at 1	—	—	ACCLO 3, ACCHI 3	—
MUL.DA.HL	mx 1, at 1	—	—	ACCLO 3, ACCHI 3	—
MUL.DA.LH	mx 1, at 1	—	—	ACCLO 3, ACCHI 3	—
MUL.DA.LL	mx 1, at 1	—	—	ACCLO 3, ACCHI 3	—
MUL.DD.HH	mx 1, my 1	—	—	ACCLO 3, ACCHI 3	—
MUL.DD.HL	mx 1, my 1	—	—	ACCLO 3, ACCHI 3	—
MUL.DD.LH	mx 1, my 1	—	—	ACCLO 3, ACCHI 3	—
MUL.DD.LL	mx 1, my 1	—	—	ACCLO 3, ACCHI 3	—
MUL16S	as 1, at 1	ar 3	—	—	—
MUL16U	as 1, at 1	ar 3	—	—	—
MULA.AA.HH	as 1, at 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULA.AA.HL	as 1, at 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULA.AA.LH	as 1, at 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULA.AA.LL	as 1, at 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULA.AD.HH	as 1, my 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—

1. The RETW and RETW.N instructions lack bypass for the AR[0] value. This is approximated by specifying that they use AR[0] in a much earlier stage (–2). See Section 10.4 on page 528.

Table 138. Xtensa T1000 Instruction Pipelining (continued)

Instruction Mnemonic	Operand Stages		Implicit State Stages		Reservations
	In	Out	In	Out	
MULA.AD.HL	as 1, my 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULA.AD.LH	as 1, my 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULA.AD.LL	as 1, my 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULA.DA.HH	mx 1, at 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULA.DA.HH.LDDEC	as 1, mx 1, at 1	mw 3, as 2	ACCLO 2, ACCHI 2, MEM 2	ACCLO 3, ACCHI 3	DataCache 2
MULA.DA.HH.LDINC	as 1, mx 1, at 1	mw 3, as 2	ACCLO 2, ACCHI 2, MEM 2	ACCLO 3, ACCHI 3	DataCache 2
MULA.DA.HL	mx 1, at 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULA.DA.HL.LDDEC	as 1, mx 1, at 1	mw 3, as 2	ACCLO 2, ACCHI 2, MEM 2	ACCLO 3, ACCHI 3	DataCache 2
MULA.DA.HL.LDINC	as 1, mx 1, at 1	mw 3, as 2	ACCLO 2, ACCHI 2, MEM 2	ACCLO 3, ACCHI 3	DataCache 2
MULA.DA.LH	mx 1, at 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULA.DA.LH.LDDEC	as 1, mx 1, at 1	mw 3, as 2	ACCLO 2, ACCHI 2, MEM 2	ACCLO 3, ACCHI 3	DataCache 2
MULA.DA.LH.LDINC	as 1, mx 1, at 1	mw 3, as 2	ACCLO 2, ACCHI 2, MEM 2	ACCLO 3, ACCHI 3	DataCache 2
MULA.DA.LL	mx 1, at 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULA.DA.LL.LDDEC	as 1, mx 1, at 1	mw 3, as 2	ACCLO 2, ACCHI 2, MEM 2	ACCLO 3, ACCHI 3	DataCache 2

1. The RETW and RETW.N instructions lack bypass for the AR[0] value. This is approximated by specifying that they use AR[0] in a much earlier stage (–2). See Section 10.4 on page 528.

Table 138. Xtensa T1000 Instruction Pipelining (continued)

Instruction Mnemonic	Operand Stages		Implicit State Stages		Reservations
	In	Out	In	Out	
MULA.DA.LL.LDINC	as 1, mx 1, at 1	mw 3, as 2	ACCLO 2, ACCHI 2, MEM 2	ACCLO 3, ACCHI 3	DataCache 2
MULA.DD.HH	mx 1, my 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULA.DD.HH.LDDEC	as 1, mx 1, my 1	mw 3, as 2	ACCLO 2, ACCHI 2, MEM 2	ACCLO 3, ACCHI 3	DataCache 2
MULA.DD.HH.LDINC	as 1, mx 1, my 1	mw 3, as 2	ACCLO 2, ACCHI 2, MEM 2	ACCLO 3, ACCHI 3	DataCache 2
MULA.DD.HL	mx 1, my 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULA.DD.HL.LDDEC	as 1, mx 1, my 1	mw 3, as 2	ACCLO 2, ACCHI 2, MEM 2	ACCLO 3, ACCHI 3	DataCache 2
MULA.DD.HL.LDINC	as 1, mx 1, my 1	mw 3, as 2	ACCLO 2, ACCHI 2, MEM 2	ACCLO 3, ACCHI 3	DataCache 2
MULA.DD.LH	mx 1, my 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULA.DD.LH.LDDEC	as 1, mx 1, my 1	mw 3, as 2	ACCLO 2, ACCHI 2, MEM 2	ACCLO 3, ACCHI 3	DataCache 2
MULA.DD.LH.LDINC	as 1, mx 1, my 1	mw 3, as 2	ACCLO 2, ACCHI 2, MEM 2	ACCLO 3, ACCHI 3	DataCache 2
MULA.DD.LL	mx 1, my 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULA.DD.LL.LDDEC	as 1, mx 1, my 1	mw 3, as 2	ACCLO 2, ACCHI 2, MEM 2	ACCLO 3, ACCHI 3	DataCache 2
MULA.DD.LL.LDINC	as 1, mx 1, my 1	mw 3, as 2	ACCLO 2, ACCHI 2, MEM 2	ACCLO 3, ACCHI 3	DataCache 2
MULS.AA.HH	as 1, at 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—

1. The RETW and RETW.N instructions lack bypass for the AR[0] value. This is approximated by specifying that they use AR[0] in a much earlier stage (–2). See Section 10.4 on page 528.

Table 138. Xtensa T1000 Instruction Pipelining (continued)

Instruction Mnemonic	Operand Stages		Implicit State Stages		Reservations
	In	Out	In	Out	
MULS.AA.HL	as 1, at 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULS.AA.LH	as 1, at 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULS.AA.LL	as 1, at 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULS.AD.HH	as 1, my 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULS.AD.HL	as 1, my 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULS.AD.LH	as 1, my 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULS.AD.LL	as 1, my 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULS.DA.HH	mx 1, at 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULS.DA.HL	mx 1, at 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULS.DA.LH	mx 1, at 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULS.DA.LL	mx 1, at 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULS.DD.HH	mx 1, my 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULS.DD.HL	mx 1, my 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULS.DD.LH	mx 1, my 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULS.DD.LL	mx 1, my 1	—	ACCLO 2, ACCHI 2	ACCLO 3, ACCHI 3	—
MULL	*not implemented*				
MULSH	*not implemented*				
MULUH	*not implemented*				
NEG	at 1	ar 2	—	—	—
NOP.N	—	—	—	—	—

1. The RETW and RETW.N instructions lack bypass for the AR[0] value. This is approximated by specifying that they use AR[0] in a much earlier stage (-2). See Section 10.4 on page 528.

Table 138. Xtensa T1000 Instruction Pipelining (continued)

Instruction Mnemonic	Operand Stages		Implicit State Stages		Reservations
	In	Out	In	Out	
NSA	as 1	at 2	—	—	—
NSAU	as 1	at 2	—	—	—
OR	as 1, at 1	ar 2	—	—	—
ORB	bs 1, bt 1	br 2	—	—	—
ORBC	bs 1, bt 1	br 2	—	—	—
POPC	as 1	at 2	—	—	—
QUOS	*not implemented*				
QUOU	*not implemented*				
REMS	*not implemented*				
REMU	*not implemented*				
RET	—	target 2	AR[0] 1	—	—
RET.N	—	target 2	AR[0] 1	—	—
RETW[1]	—	target 2	AR[0] -2	—	—
RETW.N[1]	—	target 2	AR[0] -2	—	—
RFE	—	target 2	—	—	—
RFI	—	target 2	—	—	—
RFUE	—	target 2	—	—	—
RFWO	—	target 2	—	—	—
RFWU	—	target 2	—	—	—
ROTW	—	—	—	—	—
RSIL	—	at 2	—	—	WSR 1
RSR	—	at 3	—	—	—
RSYNC	—	—	—	—	WSR 1..3
S8I	at 1, as 1	—	—	MEM 2	DataCache 2
S16I	at 1, as 1	—	—	MEM 2	DataCache 2
S32C1I	*not implemented*				
S32I	at 1, as 1	—	—	MEM 2	DataCache 2
S32I.N	at 1, as 1	—	—	MEM 2	DataCache 2
S32RI	*not implemented*				
SDCT	at 1, as 1	—	—	—	DataCache 2
SEXT	as 1	ar 2	—	—	—

1. The RETW and RETW.N instructions lack bypass for the AR[0] value. This is approximated by specifying that they use AR[0] in a much earlier stage (-2). See Section 10.4 on page 528.

Table 138. Xtensa T1000 Instruction Pipelining (continued)

Instruction Mnemonic	Operand Stages		Implicit State Stages		Reservations
	In	Out	In	Out	
SICT	at 1, as 1	—	—	—	Fetch 0..4
SICW	at 1, as 1	—	—	—	Fetch 0..4
SLL	as 1	ar 2	SAR 1	—	—
SLLI	as 1	ar 2	—	—	—
SRA	at 1	ar 2	SAR 1	—	—
SRAI	at 1	ar 2	—	—	—
SRC	as 1, at 1	ar 2	SAR 1	—	—
SRL	at 1	ar 2	SAR 1	—	—
SRLI	at 1	ar 2	—	—	—
SSA8B	as 1	—	—	SAR 2	—
SSA8L	as 1	—	—	SAR 2	—
SSAI	—	—	—	SAR 2	—
SSL	as 1	—	—	SAR 2	—
SSR	as 1	—	—	SAR 2	—
SUB	as 1, at 1	ar 2	—	—	—
SUBX2	as 1, at 1	ar 2	—	—	—
SUBX4	as 1, at 1	ar 2	—	—	—
SUBX8	as 1, at 1	ar 2	—	—	—
SYSCALL	—	—	—	—	—
UMUL.AA.HH	as 1, at 1	—	—	ACCLO 3, ACCHI 3	—
UMUL.AA.HL	as 1, at 1	—	—	ACCLO 3, ACCHI 3	—
UMUL.AA.LH	as 1, at 1	—	—	ACCLO 3, ACCHI 3	—
UMUL.AA.LL	as 1, at 1	—	—	ACCLO 3, ACCHI 3	—
WAITI	—	—	—	—	—
WSR	at 1	—	—	—	WSR 1
XOR	as 1, at 1	ar 2	—	—	—
XORB	bs 1, bt 1	br 2	—	—	—

1. The RETW and RETW.N instructions lack bypass for the AR[0] value. This is approximated by specifying that they use AR[0] in a much earlier stage (–2). See Section 10.4 on page 528.

APPENDIX D

```
#!/usr/xtensa/tools/bin/perl -w
# Generate ISA documentation from TIE files.
# $Id: GenISAHTML,v 1.6 2000/01/06 00:53:16 earl Exp $

# Copyright 1999-2000 Tensilica Inc.
# These coded instructions, statements, and computer
programs are
# Confidential Proprietary Information of Tensilica
Inc. and may not be
# disclosed to third parties or copied in any form, in
whole or in part,
# without the prior written consent of Tensilica Inc.

package Xtensa::GetISAHTML;

# Imports

# Use this to find library files
use lib $ENV{'TENSILICA_TOOLS'} . '/lib';
#use lib '@xtools@/lib';

# Perl library modules
use strict;
use Getopt::Long;

# Other modules
use HTML;
use Xtensa::TargetISA;
use Xtensa::GenISA;

# Program

# Prevent use strict errors for our global variables
use vars qw(%idiom);

{
  $::myname = 'GetISAHTML';

  # command line
  my $htmldir = undef;
  my $tpp = undef;
  die("Usage is: $::myname -htmldir dir [options...]
file\n")
    unless &GetOptions("htmldir=s" => \$htmldir,
                       "tpp=s" => \$tpp)
```

```
            && defined($htmldir)
            && @ARGV == 1;

        if (! -d $htmldir) {
          mkdir ($htmldir, 0777)
            || die("$::myname: $!, creating $htmldir.\n");
        }

        $htmldir .= '/' unless $htmldir =~ m|/$|; # ready
for catenating filenames

        my ($isafile) = @ARGV;
        my $isa = new Xtensa::TargetISA ($tpp, $isafile);

        GenISAHTML ($htmldir, $isa);
      }

      sub GenISAHTML {
        my ($htmldir, $isa) = @_;
        my $indexfh = new FileHandle ($htmldir .
'index.html', '>');
        die("$::myname: $!, opening ${htmldir}index.html for
output.\n")
          unless $indexfh;
        my $index = new HTML ($indexfh, 2);
        $index->hbegin ('Instructions');
        $index->block ('h1', 'Instructions');
        $index->block ('h2', 'Alphabetic by mnemonic');
        $index->bblock ('table');
        $index->bblock ('thead');
        $index->bblock ('tr');
        $index->block ('th', 'Mnemonic', attribute('align',
'left'));
        $index->block ('th', 'Synopsis', attribute('align',
'left'));
        $index->eblock ('tr');
        $index->eblock ('thead');
        $index->bblock ('tbody');
        my $inst;
        foreach $inst (sort {isort($a->mnemonic(), $b-
>mnemonic())}
                      $isa->instruction()) {
          my $instname = uc($inst->mnemonic());
          my $synopsis = $inst->synopsis();
          if (!defined($synopsis)) {
            print STDERR ("$::myname: No synopsis for
$instname\n");
```

```
          $synopsis = '';
        }
        $_ = $instname;
        s/\.//g;
        my $instfile = $_ . '.html';
        $index->bblock ('tr');
        $index->bblock ('th', attribute('align', 'left'));
        $index->link ($instfile, $instname);
        $index->eblock ('th');
        $index->block ('td', $synopsis, attribute('align',
'left'));
        $index->eblock ('tr');
        my $instfh = new FileHandle ($htmldir . $instfile,
'>');
        die("$::myname: $!, opening $htmldir$instfile for
output.\n")
          unless $instfh;
        my $html = new HTML ($instfh, 2);
        $html->hbegin ("$instname - $synopsis");
        instdoc ($html, $isa, $inst);
        $html->hend ();
        $html->close ();
        $instfh->close ();
      } # foreach inst
      $index->eblock ('tbody');
      $index->eblock ('table');
      $index->hend ();
      $index->close ();
      $indexfh->close ();
    }

    # Generate the instruction word box
    sub instbox {
      my ($html, $isa, $inst, $caption) = @_;
      my $instname = uc($inst->mnemonic());
      my $maxinstlen = $isa->maxsize();
      my $cellwidth = sprintf("%.0f", 720 / $maxinstlen) -
2;
      my $iv = $inst->value();
      my $im = $inst->mask();
      my $il = $inst->size();
      my $pad = $maxinstlen - $il;
      my @fields = ('') x $il;
      push (@fields, "\n");  # something to force a
mismatch
      my $oper;
      foreach $oper ($inst->operands()) {
```

```
          my $field = $oper->field();
          my $fieldname = $field->name();
          my $b;
          foreach $b ($field->bitlist()) {
            $fields[$b] = $fieldname;
          }
        }
        $html->bblock ('table', attribute('frame', 'void'),
                       attribute('rules', 'groups'),
                       attribute('cellspacing', 0),
                       attribute('cellpadding', 0));
        if (defined($caption) && $caption ne '') {
          $html->inline ('caption', $caption)
        }
        # column groups
        my $repeat;
        foreach $repeat (1 .. $pad) {
          $html->empty ('col', attribute('width',
$cellwidth));
        }
        my $j = $il-1;
        my $i;
        for ($i = $il-2; $i >= 0; $i -= 1) {
          if ($fields[$i] ne $fields[$i+1]) {
            $html->empty ('colgroup', attribute('colspan',
$j - $i));
            foreach $repeat (1 .. ($j - $i)) {
             $html->empty ('col', attribute('width',
$cellwidth));
            }
            $j = $i;
          }
        }
        $html->empty ('colgroup', attribute('colspan', $j +
1));
        foreach $repeat (1 .. ($j + 1)) {
          $html->empty ('col', attribute('width',
$cellwidth));
        }
        # bit numbers
        $html->bblock('thead');
        $html->bblock('tr');
        foreach $repeat (1 .. $pad) {
          $html->block('td', '', attribute('width',
$cellwidth));
        }
        for ($i = $il-1; $i >= 0; $i -= 1) {
```

```
          if (   $fields[$i] ne $fields[$i+1]
           || $i == 0
           || $fields[$i] ne $fields[$i-1]) {
            $html->bblock ('td', '', attribute('width',
$cellwidth),
                                attribute('align', 'center'));
            $html->inline ('small', $i);
            $html->eblock ('td');
          } else {
            $html->block ('td', '', attribute('width',
$cellwidth),
                               attribute('align', 'center'));
          }
        }
        $html->eblock ('tr');
        $html->eblock ('thead');
        # fields
        $html->bblock ('tbody');
        $html->bblock ('tr');
        if ($pad != 0) {
          $html->block ('td', '', attribute('colspan',
$pad),
                             attribute('width', $pad *
$cellwidth));
        }
        $j = $il-1;
        for ($i = $il-1; $i >= 0; $i -= 1) {
          if ($i != $j && $fields[$i] ne $fields[$i+1]) {
            $html->block ('td', $fields[$i+1],
attribute('colspan', $j - $i),
                                    attribute('width', ($j - $i)
* $cellwidth),
                                    attribute('align',
'center'),
                                    attribute('bgcolor',
'#FFE4E1'));
            $j = $i;
          }
          if ($fields[$i] eq '') {
            $b = ($iv >> $i) & 1;
            $html->block ('td', $b, attribute('width',
$cellwidth),
                               attribute('align', 'center'),
                               attribute('bgcolor', '#FFF0F5'));
            $j = $i - 1;
          }
        }
```

```
        if ($j != -1) {
          $html->block ('td', $fields[0],
attribute('colspan', $j + 1),
                                    attribute('width', ($j + 1) *
$cellwidth),
                                    attribute('align', 'center'));
        }
        $html->eblock ('tr');
        $html->eblock ('tbody');
        # field widths
        $html->bblock ('tfoot');
        $html->bblock ('tr');
        if ($pad != 0) {
          $html->block ('td', '', attribute('colspan',
$pad),
                                attribute('width', $pad *
$cellwidth));
        }
        $j = $il-1;
        for ($i = $il-2; $i >= 0; $i -= 1) {
          if ($fields[$i] ne $fields[$i+1]) {
            $html->bblock ('td', attribute('colspan', $j -
$i),
                              attribute('width', ($j - $i) *
$cellwidth),
                              attribute('align', 'center'));
            $html->inline ('small', $j - $i);
            $html->eblock ('td');
            $j = $i;
          }
        }
        $html->bblock ('td', attribute('colspan', $j + 1),
                              attribute('width', ($j + 1) *
$cellwidth),
                              attribute('align', 'center'));
        $html->inline ('small', $j+1);
        $html->eblock ('td');
        $html->eblock ('tr');
        $html->eblock ('tfoot');
        $html->eblock ('table');
      } # instbox

      # Generate documentation for instrution $inst to HTML
object $hmtl.
      sub instdoc {
        my ($html, $isa, $inst) = @_;
        my $instname = uc($inst->mnemonic());
```

```
        my $synopsis = $inst->synopsis();
        if (!defined($synopsis)) {
          print STDERR ("$::myname: No synopsis for
$instname\n");
          $synopsis = '';
        }
        $html->block ('h1', "$instname &#8212; $synopsis");
        $html->block ('h2', 'Instruction Word');
        instbox ($html, $isa, $inst);
        $html->block ('h2', 'Package');
        if ($idiom{$instname}) {
          $_ = 'Assembler Macro';
        } else {
          $_ = $inst->package();
          s/:.$//;
          my $pkglong = $pkglong{$_};
          if (defined($pkglong)) {
            $_ = $pkglong;
          } else {
            tr/a-z/A-Z/;
          }
        }
        $html->block ('p', $_);
        $html->block ('h2', 'Assembler Syntax');
        $html->bblock ('p');
        my @iasm = map ($_->name, $inst->operands);
        $html->inline ('code', @iasm == 0 ? $instname
                                  : ($instname . ' ' . join (',
', @iasm)));
        $html->eblock ('p');
        $html->block ('h2', 'Description');
        my $idesc = $inst->description();
        if (!defined($idesc)) {
          print STDERR "$::myname: No description for
$instname.\n";
        } else {
          $idesc =~ s|<INSTREF>([A-
Z.]+?)</INSTREF>|insthref($1)|gei;
          $html->iprint ($idesc);
        }
        my $iasmnote = $inst->asmnote();
        if (defined($iasmnote)) {
          $iasmnote =~ s|<INSTREF>([A-
Z.]+?)</INSTREF>|insthref($1)|gei;
          $html->block ('h2', 'Assembler Note');
          $html->iprint ($iasmnote);
        }
```

```
    $html->block ('h2', 'Operation');
    my $tiesem = $inst->tiesemantics();
    if (defined($tiesem)) {
      $html->pre ($tiesem);
    } else {
      $html->bblock ('p');
      $html->binline ('code');
      $html->iprint ('x &#8592; y');
      $html->inline ('sub', '7');
      $html->inline ('sup', '8');
      $html->iprint (' || y');
      $html->einline ('code');
      $html->eblock ('p');
    }
    $html->block ('h2', 'Exceptions');
    {
      my @exceptions = $inst->exceptions();
      if (@exceptions != 0) {
        $html->bblock ('ul');
        my $e;
        foreach $e (@exceptions) {
       my $ename = $e->name();
       my $elong = $exclong{$ename};
       $elong = $ename unless defined($elong);
       $html->block ('li', $elong);
        }
        $html->eblock ('ul');
      } else {
        $html->block ('p', 'None');
      }
    }
    my $iimpnote = $inst->impnote();
    if (defined($iimpnote)) {
      $iimpnote =~ s|<INSTREF>([A-
Z.]+?)</INSTREF>|insthref($1)|gei;
      $html->block ('h2', 'Implementation Note');
      $html->iprint ($iimpnote);
    }
  } # instdoc

  # Return HTML fragment for referencing another
instruction
  sub insthref {
    my ($inst) = @_;
    $_ = $inst;
    s/\.//g;
```

```
        '<CODE><A HREF="' . $_ . '.html">' . $inst .
'</A></CODE>';
      }

      # Local Variables:
      # mode:perl
      # perl-indent-level:2
      # cperl-indent-level:2
      # End:


      # Stuff common to GenISAHTML and GenISAMIF
      # $Id: GenISA.pm,v 1.7 1999/12/19 08:10:38 earl Exp $

      # Copyright 1999-2000 Tensilica Inc.
      # These coded instructions, statements, and computer
programs are
      # Confidential Proprietary Information of Tensilica
Inc. and may not be
      # disclosed to third parties or copied in any form, in
whole or in part,
      # without the prior written consent of Tensilica Inc.

      package Xtensa::GenISA;

      # Exports

      use Exporter ();
      @Xtensa::GenISA::ISA = qw(Exporter);
      @Xtensa::GenISA::EXPORT = qw(%pkglong %pkgchapter
%exclong &isort &generated);
      @Xtensa::GenISA::EXPORT_OK = @Xtensa::GenISA::EXPORT;
      %Xtensa::GenISA::EXPORT_TAGS = ();

      # Imports

      # Perl library modules
      use strict;

      # Module body begins here

      # Prevent use strict errors for our global variables
      use vars qw(%pkglong %pkgchapter %exclong);

      %pkglong = (
           '32bitdiv' => '32-bit Integer Divide',
           '32bitmul' => '32-bit Integer Multiply',
```

```
            'athens' => 'Xtensa V1',
            'booleans' => 'Coprocessor Option',
            'coprocessor' => 'Coprocessor Option',
            'core' => 'Core Architecture',
            'datacache' => 'Data Cache',
            'debug' => 'Debug Option',
            'density' => 'Code Density Option',
            'exception' => 'Exception Option',
            'fp' => 'Floating Point',
            'instcache' => 'Instruction Cache',
            'interrupt' => 'Interrupt Option',
            'mac16' => 'MAC16 Option',
            'misc' => 'Miscellaneous',
            'mul16' => 'Mul16 Option',
            'regwin' => 'Windowed Registers Option',
            'spec' => 'Speculation Option',
            'sync' => 'Multiprocessor Synchronization
Option',
            'timer' => 'Timer Option',
            'vectorinteger' => 'Vector Integer Coprocessor'
);
      %pkgchapter = (
            '32bitdiv' => 'ch5',
            '32bitmul' => 'ch5',
            'athens' => 'ch5',
            'booleans' => 'ch7',
            'coprocessor' => 'ch7',
            'core' => 'ch5',
            'datacache' => 'ch5',
            'debug' => 'ch5',
            'density' => 'ch5',
            'exception' => 'ch5',
            'fp' => 'ch8',
            'instcache' => 'ch5',
            'interrupt' => 'ch5',
            'mac16' => 'ch6',
            'misc' => 'ch5',
            'mul16' => 'ch5',
            'regwin' => 'ch5',
            'spec' => 'ch5',
            'sync' => 'ch5',
            'timer' => 'ch5',
            'vectorinteger' => 'vec' );
      %exclong = (
            'SystemCall' => 'System Call',
            'LoadStoreError' => 'Load Store Error',
            'FloatingPoint' => 'Floating Point Exception',
```

```
            'InstructionFetchError' => 'Instruction Fetch
Error',
            'IntegerDivideByZero' => 'Integer Divide by Zero'
);

      # Instruction name sort

      sub isort {
        my ($am, $bm) = @_;
        if ($am =~ /^([A-Za-z]+)(\d+)(.*)$/) {
          my($a1,$a2,$a3) = ($1,$2,$3);
          if ($bm =~ /^([A-Za-z]+)(\d+)(.*)$/) {
            my($b1,$b2,$b3) = ($1,$2,$3);
            return ($a1 cmp $b1) || ($a2 <=> $b2) || ($a3
cmp $b3);
          }
        }
        $am cmp $bm;
      }

      # Generated output file comment
      sub generated {
        my ($handle, $cstart, $cend, @files) = @_;
        my $date;
        chomp($date = `date`);
        $handle->print ($cstart, ' This file is
automatically generated -- DO NOT EDIT', $cend, "\n");
        $handle->print ($cstart, ' Generated from ', join('
', @files), $cend, "\n");
        $handle->print ($cstart, ' by ', $::myname, ' on ',
$date, $cend, "\n");
      }

      1;


      # Local Variables:
      # mode:perl
      # cperl-indent-level:2
      # perl-indent-level:2
      # End:
```